AFOSR-TR- 89-0428

2

T · H · E OHIO STATE UNIVERSITY

Distributed Knowledge-Based Systems

B. Chandrasekaran
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

United States Air Force
Air Force Office of Scientific Research
Bolling Air Force Base, D.C. 20332

Grant No. AFOSR-87-0090
Final Report

March 1989

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE — Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Distribution Unlimited; Available for Public Release
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): RF 765799/719026	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR-89-0428
6a. NAME OF PERFORMING ORGANIZATION: The Ohio State University Research Foundation; 6b. OFFICE SYMBOL (If applicable): OSURF	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NM
6c. ADDRESS (City, State, and ZIP Code): 1314 Kinnear Road, Columbus, Ohio 43212	7b. ADDRESS (City, State, and ZIP Code): BK1 410, BAFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: U.S. Air Force; 8b. OFFICE SYMBOL (If applicable): AFOSR/NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: Grant No. AFOSR-87-0090
8c. ADDRESS (City, State, and ZIP Code): BK1 410, Air Force Office of Scientific Research, Bolling AFB, D.C. 20332	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A7; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Distributed Knowledge-Based Systems

12. PERSONAL AUTHOR(S)
B. Chandrasekaran

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Report	FROM 11-1-86 TO 10-31-88	1989, March, 15th	320

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

→ An introduction to and survey of recent research in the generic task approach to task-specific, knowledge-based systems is presented. → [illegible]

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED/UNLIMITED] [✓ SAME AS RPT] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Major Brain Woodruff	22b. TELEPHONE (Include Area Code): (202) 767-5027; 22c. OFFICE SYMBOL: AFOSR/NM

DD Form 1473, JUN 86 — Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

THE OHIO STATE UNIVERSITY

Distributed Knowledge-Based Systems

B. Chandrasekaran
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

United States Air Force
Air Force Office of Scientific Research
Bolling Air Force Base, D.C. 20332

Grant No. AFOSR-87-0090
Final Report
RF Project 765799/719026

March 1989

Final Report under AFOSR Grant 87-0090
Distributed Knowledge-Based Systems
October 1, 1986 – October 31, 1988

B. Chandrasekaran, Principal Investigator

March 2, 1989

Recent Developments in the Generic Task Approach to Knowledge Based Systems: Introduction

Expert systems have emerged within the last decade as a result of research within artificial intelligence in problem solving and knowledge representation. Expert system technology is already found in design and diagnostic systems for a variety of domains. The first generation of expert systems attempted to make use of a uniform mode of knowledge representation and a uniform control algorithm, no matter what task was to be considered. A package of uniform representation and control strategy, such as rules with chaining or first order formulae with resolution, is often a Turing universal architecture, and as such can "in principle" carry out the computations needed for expert system problem solving. Unfortunately, universality does not by itself guarantee that a supportive environment for expert problem solving systems has been found. First, universality does not necessarily provide an efficient way to perform a problem solving task. Second, a universal architecture may result in an artificial and forced form of knowledge representation. This artificiality then means that building a system is made difficult, and also means that the ability to explain how the expert decision was made–which is important for practical expert system applications–may not be supported.

The generic task approach to knowledge based systems has from the outset taken issues of efficiency and knowledge organization to be crucial. Unlike approaches that focused on selecting a universal architecture for expert systems, the generic task approach finds task specific inference patterns and types of knowledge organization that will be useful in building expert systems for design and diagnosis. At present, numerous systems for design and diagnosis have been built, and the generic tasks of recognition, hierarchical classification, abductive assembly of hypotheses, and routine design have been used by themselves and in various combinations to achieve good performance. Explanation of expert system decisions has been supported, and the programming task of organizing the appropriate knowledge has been aided by the natural design for the generic task programming tools, now part of the Generic Task Toolset.

Knowledge based systems using task specific architectures are steadily winning recognition as significant improvements over earlier attempts at building expert systems. There are generic problem solving strategies out of which complex knowledge based reasoning systems can be built. These generic strategies make use of knowledge organized in forms that are appropriate to the task and its characteristic control regime, and in that way help encode the expert's knowledge of a domain in a deeper way than is obtained using a typical rule based system.

The research that AFOSR has supported in grant 87-0090 (which continues research previously supported by AFOSR) has developed the basic methodology of the generic task approach, has explored how deeper models and causal reasoning may be of use in problem solving systems, has helped in the production of a generic task toolset to encourage research on integration and useful in practical applications, and has examined ways in which the efficiency and robustness of systems can be improved–by learning, making use of parallelism, and by considering new environments for implementing generic tasks.

Survey of Recent Developments in the Generic Task Approach to Knowledge Based Problem Solving Systems

The progress that has been made in developing the Generic Task approach over the span of the two years of support by AFOSR grant 87-0090 can be surveyed by describing the results that have been presented in publications during that period.

Generic Tasks

We have developed the theory of generic tasks, each of which is a generic problem solving type, with characteristic types of knowledge and inference patterns [11, 2, 10]. We have shown how complex problems such as diagnosis and design can be decomposed into an interacting collection of generic tasks [14] The advantages of the Generic Task approach for knowledge acquisition have been developed in [2]. This view has also resulted in a number of high-level tools and a generic task toolkit that helps integrate these high-level tools [18, 3, 27] Our work has been pioneering in this area and has pulled the field toward task-specific problem-solving architectures.

Deep Models and Causal Reasoning

We have developed a framework for understanding the role of causal reasoning in diagnosis [8, 6], especially how to reason from structure to behavior of a system [7], how to organize the behavioral causal knowledge by using the functions of the device [5], and how to produce diagnostic knowledge from this functional-causal representation [32, 19].
Two major contributions have been the development of an approach called *consolidation* [7] for reasoning from structure to behavior and a representation called the *functional representation* [32] to encode certain kinds of causal knowledge about a system. The claim is that this representation incorporates some degree of understanding as opposed to more associational or compiled knowledge.

Generic Task Toolset

The Generic Task toolset is available to the user community. CSRL and DSPL in both Interlisp and KEE versions are currently available, and commercial versions are being marketed to the user community by a consortium headed by Battelle AI Center. A Common LISP version of the toolset is under development with support from DARPA, DEC, IBM and AFOSR [30].

Learning, Parallelism, Universal Subgoaling

Research on task specific knowledge based problem solving systems is also being directed to increase the flexibility, generality, and efficiency. A variety of approaches that make use of new architectures and implementational possiblities are being explored.

The issue of alternative (or cooperating) architectures has been investigated in a number of papers [12]. One important claim, that connectionist (or parallel distributed) architectures, like symbolic approaches, are both of interest for investigating the information processing activities of intelligence, usefully clarifies the competition between the approaches [15].

The possiblity of integrating task specific architectures with general problem solving architectures has been studied and it has been established that task specific architectures can be built in a general problem solving architecture, such as that provided by the universal subgoaling architecture of SOAR [28]. It is also likely that implementation of generic task problem solving subspaces within general problem solving systems will reduce the brittleness of the resulting problem solving systems, and ease problems of integrating generic tasks in complex systems. Brittleness may also be reduced in systems taking advantage of a rich supply of case information; integrating knowledge of cases into design systems is under investigation [21].

Abductive assembly of hypotheses is one generic task identified at LAIR [29] that has been of particular value in multiple fault diagnostic systems. Abductive assembly can be computationally expensive [1], and it is therefore of particular interest to formulate parallel algorithms for the task; versions of a parallel algorithm have been developed [26, 24].

Finally, because knowledge-based systems may fail because the knowledge they embody is defective, attempts are being made to develop systems with a capacity for corrective learning to modify their existing knowledge base when incorrect answers are produced. A procedure that compares correct explanations with the system's incorrect explanations to decide what to modify has been developed for use in a gait anomaly diagnostic system [9].

References

[1] Dean Allemang, Michael C. Tanner, Tom Bylander, and John R. Josepshson. On the computational complexity of hypothesis assembly. In *Proceedings of IJCAI-87, Milan, Italy, August, 1987*, January 1987.

[2] T. Bylander and B. Chandrasekaran. Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, 26:231–243, 1987. †

[3] T. Bylander, B. Chandrasekaran, and J. Josephson. The generic task toolset. In *Proceedings of Second International Conference on Human-Computer Interaction*, Honolulu, Hawaii, August 1987.

[4] T. Bylander, J. Smith, and J. Svirbely. Qualitative representation of behavior in the medical domain. In *Proceedings of The Fifth Conference on Medical Informatics*, pages 7–11, Washington, D.C., October 26-30 1986. Conference on Medical Informatics. †

[5] Tom Bylander. Primitives for reasoning about the behavior of devices. In *Proceedings of the AAAI Workshop on Qualitative Physics*, Urbana, Illinois, May 1987.

[6] Tom Bylander. Some causal models are deeper than others. In *Proceedings Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, July 1987. †

[7] Tom Bylander. Using consolidation for reasoning about devices. Technical report, The Ohio State University, 1987. †

[8] Tom Bylander. A critique of qualitative simulation from a consolidation viewpoint. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(2):252–268, March-April 1988. †

[9] Tom Bylander and Michael A. Weintraub. A corrective learning procedure using different explanatory types. In *Proceedings of the AAAI Mini-Symposium on Explanation-Based Learning*, Stanford University, Palo Alto, California, March 1988. †

[10] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, 1986. †

[11] B. Chandrasekaran. Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the International Joint Conference on Artificial Intelligence*. IJCAI, 1987.

[12] B. Chandrasekaran. What kind of information processing is intelligence? a perspective on ai paradigms and a proposal. In D. Partridge and Y. Wilks, editors, *Source Book on the Foundations of AI*. Cambridge University Press, 1987. †

[13] B. Chandrasekaran. Design: An information processing level analysis. In *Design Problem Solving: Knowledge Structures and Control Strategies*. Pitman: Morgan-Kaufmann, 1989. †

[14] B. Chandrasekaran. Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *Knowledge Engineering Review*, In Press 1989. †

[15] B. Chandrasekaran, A. Goel, and D. Allemang. Connectionism and information processing abstractions. In *Proceedings of the AAAI Symposium on Parallel Models of Intelligence: How Can Slow Components Think So Fast?*, pages 66–85, Stanford University, March 1988.

[16] B. Chandrasekaran and Ashok Goel. From numbers to symbols to knowledge structures: Pattern recognition and artificial intelligence perspectives on the classification task. *IEEE Transactions on Systems, Man and Cybernetics*, To appear, 1988. †

[17] B. Chandrasekaran and W. Punch III. Data validation during diagnosis, a step beyond traditional sensor validation. In *Sixth National Conference on Artificial Intelligence*. The Ohio State Univeristy, 1987. †

[18] B. Chandrasekaran, John Josephson, and David Herman. The generic task toolset: High level languages for the construction of planning and problem solving systems. In *Proceedings Workshop on Space Telerobotics*, NASA and Jet Propulsion Laboratories, Pasadena, California, 1987.

[19] B. Chandrasekaran, J. Smith, and J. Sticklen. "Deep" Models and their relation to diagnosis. In Furukawa, editor, *Medical Applications of AI*. Science Press, Amsterdam, 1987. Invited paper, Toyobo Foundation Symposium on Artificial Intelligence in Medicine, Tokyo, Japan, August 1986.

[20] Ashok Goel and Tom Bylander. Computational feasibility of structured matching. Technical report, The Ohio State University, 1989.

[21] Ashok Goel and B. Chandrasekaran. Integrating model-based reasoning and case-based reasoning for design problem solving. Technical report, The Ohio State University, 1988. †

[22] Ashok Goel and B. Chandrasekaran. Understanding device feedback. Laboratory for artificial intelligence research, The Ohio State University, Department of Computer and Information Science, March 1988. †

[23] Ashok Goel, B. Chandrasekaran, and Donald Sylvan. JESSE: an information processing model of policy decision making. In *Proceedings of the Third Annual Expert Systems in Government Conference*, pages 178–18,

Washington, D. C., October 19-23 1987. Sponsored by the IEEE Computer Society.

[24] Ashok Goel, John Kolen, and Dean Allemang. Learning in connectionist networks: Has the credit assignment problem been solved? In James Johnson, editor, *Proceedings of the Aerospace Applications of Artificial Intelligence Conference*, Dayton, Ohio, October 1988. Conference sponsored by ACM SIGART.

[25] Ashok Goel, J. Ramanujam, and P. Sadayappan. Towards a neural architecture for abductive reasoning. In *Proceedings of the Second IEEE International Conference on Neural Networks*, volume II, pages 681–688, San Diego, California, July 24-27 1988. †

[26] Ashok Goel, P. Sadayappan, and John Josephson. Concurrent synthesis of composite explanatory hypotheses. In David Bailey, editor, *Proceedings of the Seventeenth International Conference on Parallel Processing: Algorithms (nd Applications*, volume III, pages 156–160, St. Charles, Illinois, August 15-19 1988.

[27] D. Herman and D. Brown. DSPL: language for routine design. *Applied Artificial Intelligence Reporter*, 4(7):8–9,14–19, April-July 1987.

[28] Todd R. Johnson, Jr. Jack W. Smith, and B. Chandrasekaran. Generic tasks and Soar. In *AAAI Spring Symposium*, 1989. †

[29] J. Josephson, B. Chandrasekaran, J. Smith, and M. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Trans. System, Man & Cybernetics*, pages 445–454, May/June 1987. †

[30] John R. Josephson, Diana Smetters, Richard Fox, Dan Oblinger, Arun Welch, and Gayle Northrup. Integrated generic task toolset: (Fafner Release 1.0): introduction and user's guide.

[31] D.R. Myers, J.F. Davis, and D. Herman. A task oriented approach to knowledge-based systems for process engineering design. *Computers and Chemical Engineering, Special Issue on AI in Chemical Engineering Research and Development*, August 1988.

[32] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In J.L. Kolodner and C.K. Riesbeck, editors, *Experience, Memory, and Reasoning*, chapter 4, pages 47–73. Lawrence Erlbaum, 1986.

Note: the dagger, †, indicates that the item is included in the following appendix of articles that is a selection of research supported under AFOSR Grant 87-0090.

Attachments

RF Project 765799/719026
Technical Report

GENERIC TASKS FOR KNOWLEDGE-BASED REASONING: THE "RIGHT" LEVEL OF ABSTRACTION FOR KNOWLEDGE ACQUISITION

Tom Bylander and B. Chandrasekaran
Department of Computer and Information Science

For the Period
December 1986

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Building 410
Bolling AFB, D. C. 20332-6448

Contract No. AFOSR-87-0090

June 1987

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
December 1986

GENERIC TASKS FOR KNOWLEDGE-BASED REASONING: THE "RIGHT" LEVEL OF ABSTRACTION FOR KNOWLEDGE ACQUISITION

Tom Bylander and B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

To appear in *International Journal of Man-Machines Studies.*

Generic Tasks for Knowledge-Based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition[*]

Tom Bylander and B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio, USA 43210

Abstract

Our research strategy has been to identify *generic tasks* -- basic combinations of knowledge structures and inference strategies that are powerful for dealing for certain kinds of tasks. Our strategy is best understood by considering the "interaction problem," that the representation of knowledge strongly interacts with the inference strategy that is applied to the knowledge and with the task that the knowledge is used for. The interaction problem implies that different knowledge acquisition methodologies will be required for different kinds of reasoning, e.g., a different knowledge acquisition methodology for each generic task. We illustrate this using the generic task of hierarchical classification. Our proposal and the interaction problem call into question many generally held beliefs about expert systems, such as the belief that the knowledge base should be separated from the inference engine.

Introduction

Knowledge acquisition is the process that extracts knowledge from a source (e.g. a domain expert or textbook) and incorporates it into a knowledge-based system. Because a knowledge-base representation is its target, knowledge acquisition cannot be separated from a broader theory of knowledge-based reasoning. A solution to knowledge acquisition must be compatible with a solution to the general problem of knowledge-based reasoning.

For some time now, we have been developing a *theory of generic tasks* that identifies several types of reasoning that knowledge-based systems perform and provides a overall framework for the design and implementation of such

[*]Research supported by Air Force Office of Scientific Research, grant 82-0255, National Science Foundation grant MCS-8305032, and Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010.

systems [Chandrasekaran 83, Chandrasekaran 84, Chandra 86 86]. In this paper, we present our theory as a way to exploit the interaction problem. Because each generic task exploits it differently, each one should be be associated with a different knowledge acquisition methodology.

First, we pose and discuss the "interaction problem." Next, we review our theory of generic tasks: the characteristics of a generic task and the generic tasks that have been identified so far. In view of the interaction problem, we propose our theory of generic tasks as a framework for identifying different knowledge acquisition methodologies. We illustrate this using the generic task of hierarchical classification. Finally, we reflect on a number of beliefs that have driven much of the past research on knowledge acquisition and knowledge-based reasoning.

The Interaction Problem

The interaction problem is this:

> *The representation of knowledge strongly interacts with the inference strategy that is applied to the knowledge and with the task that is performed with it.*

In other words, knowledge representations have a close relationship to how they are used to solve problems. Knowledge is dependent on its use. The interaction problem is not a new concept. Minsky, in his famous frame proposal, argues that "factual and procedural contents must be more intimately connected to explain the apparent power and speed of mental activities" [Minsky 75 - p. 211]. Marr has noted that "how information is represented can greatly affect how easy it is to do different things with it" [Marr 82 - p. 21]. Our argument takes a different perspective, that the inference strategy and task influence what knowledge that is represented, i.e., we will represent knowledge so that it takes advantage of how we are going to use it.

The interaction problem, if true, has serious implications for how knowledge acquisition should be done. Because some knowledge representation must be the target of knowledge acquisition, knowledge acquisition methodologies must take the interaction problem into account. Also, if different kinds of reasoning have different kinds of interactions, there is a need for a different knowledge acquisition methodology for each kind of reasoning.

The Interaction Problem for General Knowledge Representations

Each of the major forms of knowledge representation -- rules, logic, and frames -- are subject to the interaction problem. Knowledge is affected by the task (different tasks require different knowledge) and by inference strategy (knowledge is adapted to the strategy). In addition to giving examples of the interaction problem, we also wish to emphasize that the generality of a representation does not make the interaction problem disappear.

The Interaction Problem in Rules. The idea of rules is to explicitly map situations to actions. Naturally then, the focus is on determining what conditions characterize the situations and what conclusions characterizes the actions. The result is that two different tasks in the same domain can have different rules representing the "same" knowledge. For example in diagnosis, rules of the form "symptom --> malfunction" will be implemented, while in prediction of symptoms, the rules will be in the form "malfunction --> symptom". In each case, the rules will be tuned to the task that is being performed. One might argue that there is no problem with keeping both tasks in mind, and so both kinds of rules can be implemented at the same time. Of course, *given that one has already taken the interaction problem into account*, the knowledge base then will have rules appropriate for the tasks to be performed.

Another source of interaction is that special programming techniques are needed to encode task-specific inference strategies. For example, R1 [McDermott 82], which is implemented in OPS5 [Forgy 81], performs a sequence of design subtasks, each of which is implemented as a set of production rules. However, OPS5 has no construct equivalent to a subtask, so the grouping of rules and the sequencing from one set of rules to another are achieved by programming techniques. Clearly, R1's task structure has had an significant effect on how knowledge was encoded in OPS5's production rule representation.

Different inference strategies for rules are also a source of interaction. If EMYCIN's backward-chaining strategy is used, rules can combine with other rules to increase or decrease confidence in a given conclusion [van Melle 79]. On the other hand, if OPS5's recognize-and-act strategy is used, only one rule at a time can fired, so that situations must be matched to actions much more exactly. Also, the "context" must be carefully controlled to ensure that appropriate rules are considered. Note that the difference is not whether EMYCIN does forward- or backward-chaining, but that EMYCIN allows rules to act in parallel, while OPS5 applies rules in serial.

The Interaction Problem in Logic. Rule-based and logic-based representations are fairly similar with respect to the interaction problem. Like rules, logic provides for a direct way for drawing conclusions from situations. In the context of a specific task, it is only useful to encode propositions that can make task-relevant conclusions. Propositions for a diagnostic system would be like "if symptom A, then maybe malfunctions X or Y or Z," while a prediction system would have propositions like "if malfunction X, then maybe symptoms A or B or C." This example also shows one danger in applying both kinds of propositions indiscriminately. Given some confidence in malfunction X, then some confidence in symptom A should be inferred, followed by inferring some confidence in malfunctions Y and Z, which probably wasn't intended.

Logic-based representations are also like rules with respect to implementing the structure of a task and dealing with different inference strategies. To implement R1 in predicate logic, for example, a subtask construct would also have to be implicitly programmed. Two different inference strategies for logic, such as PROLOG and resolution theorem proving, are quite different to use.

The Interaction Problem in Frames. The emphasis in frame representations is on describing the conceptual structure of the domain. However, different tasks might need quite different conceptual structures. For example, classificatory problem solving [Gomez and Chandrasekaran 81, Clancey 85] in general needs a generalization hierarchy (hypothesis-subhypothesis), while routine design [Brown and Chandrasekaran 84] in general needs a structural hierarchy (component-subcomponent).

Frames are intended to flexibly interact with the inference strategy. After all, the idea of procedural attachment is to embed procedures in the frames so that the appropriate inferences are triggered.

Exploiting the Interaction Problem

The interaction problem will not go away no matter what representation is chosen. Every knowledge-based system will be developed, debugged, and maintained so its knowledge works with its inference strategy and so its knowledge in combination with its inference strategy solves a certain set of problems. No one undertakes an exhaustive study of a domain, i.e., acquires any and all the knowledge associated with that domain. It would take too long and we humans are too lazy anyway. We have learned that only a selected portion of domain knowledge needs to be acquired to perform specific tasks.

Instead of trying to lessen the impact of the interaction problem, our research strategy has been to *exploit* it. Our strategy is not new; exploiting the interaction problem has been the "untold story" of knowledge-based systems, and perhaps of AI in general. This should be obvious to anybody who has ever maintained a knowledge-based system. To find and correct an error, one has to understand both the problem solving and the knowledge base, and how they combined to cause the error.

It should not be surprising that different representations can be exploited in different ways and are thus more applicable to certain kinds of tasks than others. This is where our theory of generic tasks comes in. Our intent is to propose kinds of reasoning in which the representation and the inference strategy can be exploited to solve certain kinds of tasks. For a particular domain and task, our intent is to encode a selected portion of domain knowledge into an efficient and maintainable problem solving structure.

The Proposal

Intuitively one would think that diagnosis in different domains would have certain types of reasoning in common, and that design in different domains would also have certain types of reasoning in common, but that diagnostic reasoning and design problem solving will be generally speaking different. For example, diagnostic reasoning generally involves malfunction hierarchies, rule-out strategies, setting up a differential, etc., while design involves device/component hierarchies, design plans, ordering of subtasks, etc. However, the formalisms (or equivalently the languages) that have been commonly used for knowledge-based systems do not capture these distinctions. Ideally, diagnostic knowledge should be represented by using the *vocabulary* that is appropriate for diagnosis, while design knowledge should have a vocabulary appropriate for design. Our approach to this problem has been to identify *generic tasks* -- basic combinations of knowledge structures and inference strategies that are powerful for dealing for certain kinds of tasks. The generic tasks provide a vocabulary for describing reasoning tasks, as well as for designing knowledge-based systems that perform them.

Characterization of a Generic Task

Each generic tasks is characterized by information about the following:

1. The type of *task* (the type of input and the type of output). What is the function of the generic task? What is the generic task good for?

2. The *representation* of knowledge. How should knowledge be organized and structured to accomplish the function of the generic task?

3. The *inference strategy* (process, problem solving, control regime). What inference strategy can be applied to the knowledge to accomplish the function of the generic task?

4. The type of *concepts* that are involved in the generic task. What concepts are the input and output about? How is knowledge organized in terms of concepts? How does the inference strategy operate on concepts? In essence, we adopt Minsky's idea of frames as a way to organize the problem solving process [Minsky 75].

The phrase "generic task" is somewhat misleading. What we really mean is *an elementary generic combination of a task, representation, and inference strategy about concepts.* The power of this proposal is that if a problem matches the function of a generic task, then the generic task provides a knowledge representation and an inference strategy that can be used to solve the problem. It should be noted that a problem might match the function of more than one generic task, so that several strategies might be used to solve the problem, depending on the knowledge that is available. Also, generic tasks can be composed for more complex reasoning, i.e., a generic task may call upon another generic task to solve a subproblem.

Examples of Generic Tasks

Our group has identified several generic tasks. Here, we briefly describe the generic tasks of hierarchical classification [Gomez and Chandrasekaran 81] and object synthesis by plan selection and refinement [Brown and Chandrasekaran 84].

Hierarchical Classification. Task: *Given a description of a situation, determine* what categories or hypotheses apply to the situation.

Representation: The hypotheses are organized as a classification hierarchy in which the children of a node represent subhypotheses of the parent. There must be knowledge for calculating the degree of certainty of each hypothesis.

Inference Strategy: The establish-refine strategy specifies that when a hypothesis is confirmed or likely (the establish part), its subhypothesis should be considered (the refine part). Additional knowledge may specify how refinement is performed, e.g. to consider common hypotheses before rarer ones. If a hypothesis is rejected or ruled-out, then its subhypotheses are also ruled-out.

Important Concepts: Hypotheses.

Examples: Diagnosis can often be done by classification. In planning, it is often useful to classify a situation as a certain type, which then might suggest an appropriate plan. MYCIN [Shortliffe 76] can be thought of as classifying a patient description into an infectious agent hierarchy. PROSPECTOR [Duda *et al.* 80] can be viewed as classifying a geological description into a type of formation.

Object Synthesis by Plan Selection and Refinement. Task: Design an object satisfying specifications. An object can be an abstract device, e.g. a plan or program.

Representation: The object is represented by a component hierarchy in which the children of a node represent components of the parent. For each node, there are plans that can be used to set parameters of the component and to specify additional constraints to be satisfied. There is additional knowledge for selecting the most appropriate plan and to recover from failed constraints.

Inference Strategy: To design an object, *plan selection and refinement* selects an appropriate plan, which, in turn, requires the design of subobjects at specified points in time. When a failure occurs, failure handling knowledge is applied to make appropriate changes.

Important Concepts: The object and its components.

Examples: Routine design of devices and the synthesis of everyday plans can be performed using this generic task. The MOLGEN work of Friedland [Friedland 79] can be viewed in this way. Also R1's subtasks [McDermott 82] can be understood as design plans.

Other Generic Tasks

Other generic tasks that have been identified include knowledge-directed information retrieval [Mittal *et al.* 84], abductive assembly of explanatory hypotheses [Josephson *et al.* 84], hypothesis matching [Chandrasekaran *et al.* 82], and state abstraction [Chandrasekaran 83]. More detail on the overall framework can be found in [Chandra 86 86].

Exploiting Classificatory Problem Solving

Each generic task exploits domain knowledge in a different way; it calls for knowledge in a specific form that can be applied in a specific way. Because the knowledge acquisition methodology must be able to extract and select the appropriate knowledge, each generic tasks calls for a different knowledge acquisition methodology. For illustration we consider the generic task of hierarchical classification. In classification, the emphasis is on obtaining the classification hierarchy that contains the hypotheses that are relevant to the task and adaptable to the strategy. This section does not provide a complete knowledge acquisition methodology for classification, but outlines a number of considerations that a methodology must take into account. Additional guidelines for using classification can be found elsewhere [Mittal 80, Bylander and Smith 85].

Determining Hypotheses of Interest

Classificatory problem solving is useful for determining the hypotheses that apply to a situation. The first step then is to decide upon the hypotheses that the problem solver should potentially output. For example in diagnosis, the potential malfunctions of the object should be considered. The goal here is to determine the specific categories that should be produced, so if a general category is considered (e.g. "something is wrong with X"), then more specific categories should be generated (e.g. by asking "What types of problems can occur with X?"). Determining the usefulness of a category is discussed below.

Analyzing Commonalities among Hypotheses

Once a collection of classificatory hypotheses have been identified, one needs to determine the commonalities among the hypotheses. These commonalities become potential candidates for mid-hierarchy hypotheses in the classification hierarchy.

The easiest example to handle is when one hypothesis is clearly a subhypothesis of another i.e., it asserts a more specific category. In general, two hypothesis may have commonalities along the following lines:

- Definitional - The two hypotheses share a definitional attribute, e.g., hepatitis and cirrosis are liver diseases. Rain and snow are forms of precipitation.

- Appearance - The two hypotheses are recognized using common pieces of evidence. Both cholestasis and hemolytic anemia have jaundice as a common symptom. Wet grass is symptomatic of both rain and dew.

- Planning - The two hypotheses are associated with similar plans of action. Both the common cold and allergies are reasons to take plenty of facial tissue with you. Either lightning or strong winds are good reasons for staying inside.

The ideal hypothesis asserts some definitional attribute over all its subhypotheses, has an appearance common to all its subhypotheses, and also provides constraints on the plans associated with its subhypotheses.

In general, the hierarchy should follow a definitional decomposition whenever possible. However, there are cases where appearance is an important consideration. For example, the Dubin-Johnson syndrome is a benign hereditary disorder that mimics key symptoms of cholestasis (jaundice, conjugated hyperbilirubinemia - high amounts of conjugated bilirubin in the blood). Because it *looks* so much like cholestasis, it is most useful to make it a subhypothesis of cholestasis.

Assessing Evidence for or against Hypotheses

The above two steps should generate a large number of hypotheses. However, not all of them will be useful for classificatory problem solving, i.e., there is a need to select a classification hierarchy that can be used to exploit the establish-refine strategy, getting rid of any intermediate hypothesis do not provide additional problem solving power. Because the language we have used for classification, CSRL, requires a classification *tree* [Bylander and Mittal 86], we have become familiar with some of the strategies for evaluating hypotheses. However, the following questions are relevant whether a tree or tangled hierarchy is used.

- Are there sufficient criteria to distinguish the hypothesis from other hypotheses? In other words, does this hypothesis have a different appearance from other hypotheses?

- Is there evidence that distinguishes the hypotheses from its siblings? Because the establish-refine strategy does not consider a hypotheses unless its parent (or one of its parents in a tangled hierarchy) is relevant, evidence that distinguishes the hypothesis from its siblings is especially important.

- Is the evidence normally available? Evidence for or against an hypothesis is not very useful if is not likely to be available to the system when it is running. For example in medical diagnosis, some tests are relatively risky, expensive, or time-consuming to perform, so it is best to use hypotheses that rely on outward signs and symptoms and generally available laboratory data.

We have generally used another generic task, hypothesis matching, for mapping evidence to confidence values in hypotheses [Chandrasekaran *et al.* 82]. However, we do not want to complicate the central issue by considering combinations of generic tasks. Examples of how hypothesis matching can be exploited are provided in [Sticklen *et al.* 85] and [Bylander and Mittal 86].

Debugging Hypotheses

An important part of knowledge acquisition is being able to find out what knowledge was incorrect or left out when something goes wrong. In classification, the following problem can occur.

- Wrong confidence value - Debug the knowledge that produces the confidence value. Sticken *et al.* [Sticklen *et al.* 85] describes how hypothesis matching can be debugged. The problems below assume that the confidence values are reasonable in view of the evidence considered.

- Bad hierarchical structure - If a hypothesis was incorrectly considered or incorrectly left unconsidered, the problem may be simply that the hypothesis is in the wrong place in the hierarchy. The hypothesis is not, by definition or appearance, a subhypothesis of its parent (or some other ancestor).

- Failure to consider - Another reason why a hypothesis might not be considered is because an ancestor was not refined. The problem might be that there isn't enough evidence to support a high level of confidence in the ancestor. In this case, a restructuring is necessary in order to provide a way for establish-refine to reach the hypothesis. Less drastic solutions are lowering the threshold for refining the ancestor or considering more evidence for the ancestor.

- Establish-refine strategy is too simple - Sometimes a hypothesis should not be considered even if its parent is established. For example, if one of the hypothesis's siblings is confirmed, and the hypothesis is incompatible with its siblings, then the hypothesis should not be considered. The solution here is o adapt the establish-refine strategy to take this additional information into account. It should be noted that this problem is not a defect of establish-refine. Instead, it shows that establish-refine is really a *family* of strategies. The CSRL language, for example, provides a default establish-refine strategy and allows other establish-refine strategies to be defined.

The Point of Classification

The real reason for doing hierarchical classification is to obtain the hypotheses that describe a situation. Therefore, we should adopt knowledge acquisition methodologies that are intended to produce efficient and maintainable classification systems. To do this, we need to exploit the interactions between the representation, inference strategy, and the task. For classification, it means that each potential hypotheses of a classification hierarchy must be evaluated with respect to how well it interacts with the establish-refine strategy and whether the task may need to output it.

A Reexamination of Past Beliefs

Some generally held beliefs about knowledge-based systems need to be reexamined in light of the interaction problem and our proposal to exploit it. These beliefs have served the first generation of knowledge-based systems well, especially in stimulating much research and discussion. However, we believe it is the time to reconsider them.

Belief #1: Knowledge should be uniformly represented and controlled.

This belief denies the interaction problem and implies that there is nothing to be gained by using different representations to solve different problems. Our experience is that when the problems of a domain match the generic tasks, the generic tasks provide explicit and powerful structures for understanding and organizing domain knowledge.

Belief #2: The knowledge base should be separated from the inference engine.

This belief denies that the inference strategy affects how knowledge is represented. However, its real effect has been to force implementors to implicitly encode inference strategies within the knowledge base. Both MYCIN, whose diagnostic portion is best understood as classification, and R1, which is best understood as routine design, show that this separation is artificial.

Belief #3: Control knowledge should be encoded as metarules.

Although metarules address the problem of how to have multiple, explicit strategies in a rule-based system, the metarule approach ignores other aspects of the interaction problem. The "separation of control knowledge from domain knowledge" promotes the view that domain knowledge can be represented independent of its use, i.e., that different sets of metarules can be applied as needed. However, given a clear strategy (whether metarules or inference engine) and a task to be performed, the domain knowledge will be adapted to interact with the strategy to solve the task.

Belief #4: The ontology of a domain should be studied before considering how to process it.

We believe that ontology should not be performed just for its own sake, but in view of the tasks that need to be done. For example, to apply classification to a domain, there is a need to focus on the hypothesis space and evaluate hypotheses. Although other knowledge structures (e.g. component hierarchies, causal networks) may be useful for other generic tasks, if classification is going to be performed, then knowledge acquisition should concentrate on those aspects of the domain that are relevant to classification. This is not to say that a domain shouldn't be analyzed to identify what generic tasks are appropriate; however, this kind of domain analysis does not require an exhaustive ontology of the domain.

Belief #5: Correct reasoning is a critical goal for knowledge-based systems.

Everything else being equal, being correct is better than being incorrect. However, an emphasis on cotrectness detracts from more critical issues. One of those issues is developing an understanding of the appropriate strategies to be applied to a task. For example, there has been much research and debate about normative methods for calculating uncertainty. The reasoning problem, though, is not how to precisely calculate uncertainty, but how to avoid doing so. In diagnosis, e.g., there is much more to be gained by using abduction (assembling composite hypotheses to account for symptoms), then by independently calculating the degree of certainty of each hypothesis to several decimal places of accuracy.

Belief #6: Completeness of inference is a critical goal for knowledge-based systems.

Everything else being equal, being complete is better than being incomplete, but an emphasis on completeness ignores the fact that certain kinds of inferences will be more important than others for a particular task. For example in our description of classification, we did not mention that when a subhypothesis is confirmed, one can infer that its ancestors are also confirmed. However, that inference is not the crucial aspect of classification. The important inference is that when a hypothesis is confirmed or likely, then its subhypotheses should be considered.

Belief #7: A representation that combines rules, logic, frames, etc. is what is needed.

Such representations appear to be a good compromise since they let you represent knowledge in the "paradigm" of your choice. Unfortunately, this is, at best, only an interim solution until something better is found. None of the individual representations fully address the interaction problem, nor do they distinguish between different types of reasoning.

Generic Tasks at the "Right" Level of Abstraction

The first generation of research into knowledge-based systems has conducted a extensive search for a "holy grail" of representation, in which knowledge could be represented free of assumptions of how it would be used. For any particular task, though, certain kinds of inferences will become critical to the task, and consequently, domain knowledge needs to be organized so those inferences are performed efficiently. This is how the interaction problem arises, and why it will never go away. Instead of futilely trying to avoid it, the interaction problem needs to be studied and understood so that methods of exploiting it can be discovered and applied.

Our theory of generic tasks is an attempt to provide the "right" level of abstraction for this and other problems of knowledge-based reasoning. Each generic task provides a knowledge structure in which knowledge can be organized at a conceptual level. In classification, the concepts are hypotheses organized as a classification hierarchy. Each generic task identifies a combination of a task definition, representation, and inference strategy that exploits the interaction problem. We have shown how the generic task of classification can be associated with a knowledge acquisition methodology that takes advantage of the interactions between domain knowledge and classificatory problem solving.

References

[Brown and Chandrasekaran 84]
D. C. Brown and B. Chandrasekaran.
Expert Systems for a Class of Mechanical Design Activity.
In *Proc. IFIP WG5.2 Working Conference on Knowledge Engineering in Computer Aided Design.* IEEE Computer Society, Budapest, Hungary, 1984.

[Bylander and Mittal 86]
T. Bylander and S. Mittal.
CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling.
AI Magazine 7(2):66-77, 1986.

[Bylander and Smith 85]
T. Bylander and J. W. Smith.
Mapping Medical Knowledge into Conceptual Structures.
In *Proc. Expert System in Government Symposium*, pages 503-511. IEEE Computer Society, McLean, Virginia, 1985.

[Chandra 86 86] B. Chandrasekaran.
Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design.
IEEE Expert 1(3):23-30, 1986.

[Chandrasekaran 83]
B. Chandrasekaran.
Towards a Taxonomy of Problem Solving Types.
AI Magazine 4(1):9-17, 1983.

[Chandrasekaran 84]
B. Chandrasekaran.
Expert Systems: Matching Techniques to Tasks.
Artificial Intelligence Applications for Business.
Ablex, Norwood, New Jersey, 1984, pages 116-132.

[Chandrasekaran *et al.* 82]
B. Chandrasekaran, S. Mittal, and J. W. Smith.
Reasoning with Uncertain Knowledge: The MDX Approach.
In *Proc. Congress of American Medical Informatics Association*, pages 335-339. AMIA, San Francisco, 1982.

[Clancey 85] W. J. Clancey.
Heuristic Classification.
Artificial Intelligence 27(3):289-350, 1985.

[Duda *et al.* 80] R. O. Duda, J. G. Gaschnig, and P. E. Hart.
Model Design in the Prospector Consultant System for Mineral Exploration.
Expert Systems in the Microelectronic Age.
Edinburgh University Press, 1980, pages 153-167.

[Forgy 81] C. L. Forgy.
OPS5 Users Manual.
Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.

[Friedland 79] P. Friedland.
Knowledge-based Experiment Design in Molecular Genetics.
PhD thesis, Computer Science Department, Stanford University, 1979.

[Gomez and Chandrasekaran 81]
F. Gomez and B. Chandrasekaran.
Knowledge Organization and Distribution for Medical Diagnosis.
IEEE Trans. Systems, Man and Cybernetics SMC-11(1):34-42, 1981.

[Josephson *et al.* 84]
J. R. Josephson, B. Chandrasekaran, and J. W. Smith.
Assembling the Best Explanation.
In *Proc. IEEE Workshop on Principles of Knowledge-Based Systems*. pages 185-190. IEEE Computer Society, Denver, 1984.

[Marr 82] D. Marr.
Vision.
W. H. Freeman, 1982.

[McDermott 82] J. McDermott.
R1: A Rule-based Configurer of Computer Systems.
Artificial Intelligence 19(1):39-88, 1982.

[Minsky 75] M. Minsky.
A Framework for Representing Knowledge.
The Psychology of Computer Vision.
McGraw-Hill, 1975, pages 211-277.

[Mittal 80] S. Mittal.
Design of a Distributed Medical Diagnosis and Database System.
PhD thesis, Dept. of Comp. and Info. Sci., The Ohio State University, 1980.

[Mittal *et al.* 84]
S. Mittal, B. Chandrasekaran, and J. Sticklen.
Patrec: A Knowledge-Directed Database for a Diagnostic Expert System.
Computer 17(9):51-58, 1984.

[Shortliffe 76] E. H. Shortliffe.
Computer-Based Medical Consultations: MYCIN.
Elsevier, New York, 1976.

[Sticklen *et al.* 85]
J. Sticklen, B. Chandrasekaran, and J. W. Smith.
MDX-MYCIN: The MDX Paradigm Applied to the MYCIN Domain.
Computers and Mathematics with Applications 11(5):527-539, 1985.

[van Melle 79] W. van Melle.
A Domain Independent Production-Rule System for Consultation Programs.
In *Proc. Sixth Interational Conf. on Artificial Intelligence*, pages 923-925. Tokyo, 1979.

86-TB-QUALIT

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
November 1986

QUALITATIVE REPRESENTATION OF BEHAVIOR IN THE MEDICAL DOMAIN

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science

Jack Smith and John Svirbely
Laboratory for Knowledge-Based Medical Systems
Department of Pathology

The Ohio State University
Columbus, OH 43210

Appears in *Proc. The Fifth Conference on Medical Informatics (MEDINFO 86)*, pages 7-11. Washington D.C., October 26-30, 1986.

Qualitative Representation of Behavior in the Medical Domain

Tom Bylander, Jack W. Smith, Jr., M.D.*, and John R. Svirbely, M.D.*

Laboratory for Artificial Intelligence Research and *Laboratory for Knowledge-Based Medical Systems
Department of Computer and Information Science and *Department of Pathology
The Ohio State University
Columbus, Ohio 43210 USA

There is a need for models of the human body which can predict and explain behavior based on inexact information and on changes in structure and behavior. We present a framework for representing the structure and behavior of physical systems and apply it to the human cardiovascular system. Our framework allows for hierarchical representation of physical systems, and facilitates two kinds of reasoning processes: composition of behaviors and qualitative simulation.

1. Need for Model-Based and Qualitative Reasoning in the Medical Domain

Most knowledge-based programs depend upon *compiled* knowledge, i.e., their knowledge associates data with conclusions, but does not represent the causal relationships among the data and conclusions. For example, MYCIN [10] associates the datum "head injury" with a certain amount of confidence in the conclusion "E. Coli causing meningitis," but MYCIN does not have knowledge which explains this association, in this case, how head injury can result in E. Coli causing meningitis. In addition, MYCIN would need an exhaustive set of rules in order to block associations *when they are not appropriate* (e.g., *if the head injury* occurred in a sterile environment), and to give weight to conclusions in causally similar circumstances (e.g., if other pathways to the meninges are available to E. Coli). Examples like these are not just true for MYCIN, but occur whenever compiled knowledge is not supported by causal knowledge.

One approach to this problem is to develop a representation which models the domain and permits the use of reasoning processes to predict and explain behavior. In a medical setting, such an approach needs to work under the following constraints:

- Incomplete information. Complete and precise data on a patient are not available.
- Variability. The model should be able to accommodate normal variability due to factors such as age, sex, race, etc.
- Abnormality. It is expensive to develop different models for each possible disease and combination of diseases. It would be better to develop a single model which can be readily modified to reflect abnormal situations.

Current quantitative models can do prediction accurately, but are poor at satisfying the above constraints and at causal explanation. Our research explores the use of qualitative reasoning on hierarchical representations of the behavior and structure of the human body. Qualitative reasoning can be performed in the context of incomplete information and large ranges of variability, while still providing substantial predictive and explanatory power. We present a general framework for representing physical systems, and apply it to the cardiovascular system. Our primary concern is to show how causal knowledge can be adequately represented.

2. Why Hierarchical Representations of Structure and Behavior

Any model of the human body needs to consider the following two problems:

- Changes in the body's physical structure may change the body's behavior.
- Changes in the behavior of a body part may change the body's behavior.

To handle these cases, a representation needs to express the structure and behavior of the human body, and reasoning processes need to be able to take this information into account.

Another problem is that the multitude of body parts makes it difficult to effectively reason about its overall behavior, necessitating a hierarchical representation. In our framework, the hierarchy is structural: the body and its systems can be described as a configuration of smaller systems, each with its own behavioral description. Advantages of a hierarchically organized representation are that more detailed knowledge is not needed until it is relevant, and that the representation can be checked by determining whether the behavior of a system is consistent with the behavior of its constituents.

3. A Behavioral Representational System

In this section, we briefly describe the behavioral representation that Bylander and Chandrasekaran have been developing [1], and illustrate its application to the cardiovascular system. This representation facilitates a process called *consolidation* for efficiently inferring the behavior of a system from the behavior and structure of its elements. The major processing sequence of consolidation is to hypothesize a composite component consisting of two other components, and then to infer the behavior of the composite from the behavioral description of the components. Successful application of this sequence on increasingly larger composite components results in inferring the behavior of the whole system and in explaining how the overall behavior is caused by the components' behavior.

The representation can also be used to predict behavior. Given initial conditions and outside interactions, qualitative simulation can be performed in a manner similar to current proposals [2, 3, 5].

3.1. Elements

A physical system consists of objects which cause it to behave in a certain way. As a first approximation, these objects can be divided into two classes:

- Components. These form the physical structure of the system. Veins, arteries, and heart chambers and valves are examples of components.
- Substances. These move between the components of the system, and are what the components act upon. The term "substance" is intended to also include non-material phenomena that are commonsensically thought of as substances, such as heat and light. Blood and nervous signals are other examples of substances.

Structural relationships among components and substances are of two general types:

- Connection between components. A connection implies that the components are next to each other spatially, and that certain kinds of substances can pass through the connection.
- Containment of substances. Both components and substances may contain substances. For example, arteries contain blood, and blood contains oxygen.

Figure 1 illustrates our representation of the cardiovascular system's structure. Each box in the figure represents a component; lines between boxes represent connections. `Cardiovascular Control` represents that part of the nervous system which regulates and synchronizes the other components. Open connections indicate where the cardiovascular system interacts with other systems of the body. Only two open connections, `lung connection` and `interstitial connection`, are shown. All the components except for `Cardiovascular Control` contain blood. Blood contains a number of substances, including oxygen and carbon dioxide.

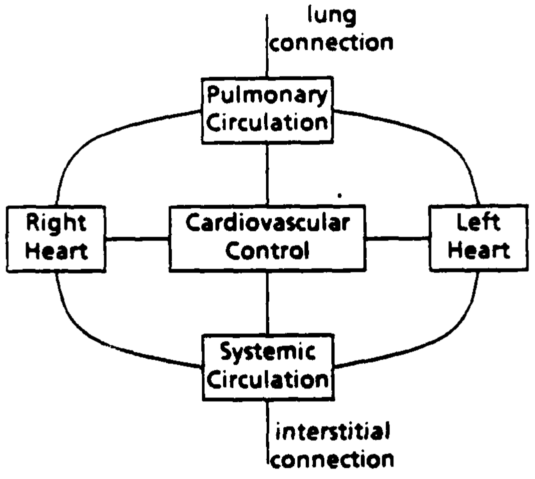


Figure 1: Components of the Cardiovascular System

3.2. Behaviors

There are two classes of behavior: actions upon components and actions upon substances. Actions upon substances consist of the following types:

- *Allow.* The object permits a specified kind of substance to move from one place to another. A vein has an *allow* blood behavior.
- *Influence.* The object tries to move a specified kind of substance. There are two subtypes according to the spatial relationship of the *influence* to the structure.
  - *Expel.* The object tries to move a substance from (or to) an internal container. The chambers of the heart have *expel* blood behaviors.
  - *Pump.* The object tries to move a substance through some path. The heart has a *pump* blood behavior.
- *Move.* The object moves a specified kind of substance from one container to another along a specified path. The cardiovascular system has a *move* blood behavior. *Move* behaviors are implicitly constrained by the amount and capacity of the containers.
- *Create.* The object creates a specified kind of substance in a container. The nervous system has a *create* signal behavior. Bone marrow (which we will not model here) has a *create* red blood cell behavior.
- *Destroy.* The object destroys a specified kind of substance in a container. A mononuclear cell has a *destroy* senescent red blood cell behavior. A transformation of a substance can be modeled by a combination of *destroy* and *create* behaviors.

Each type of behavior is associated with quantities which specify the attributes of the behavior, e.g., resistance of an *allow* behavior, amount of an *influence*, and rate of a *move* or *create*.

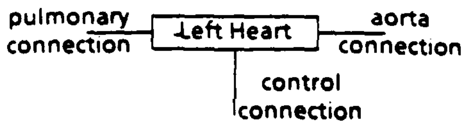
There is one kind of action upon components, *change of state*, in which a component changes state under certain conditions, e.g., when some behavior occurs or some time limit is reached. A behavior of a component may be active only under certain states.

The quantities of a behavior may depend on other behaviors occurring. For example, the amount of the heart's *pump* blood behavior depends on signals from the nervous system, i.e., *move* signal behaviors.

A very simple model of the cardiovascular system (figure 1) would attribute *pump* blood behaviors to `Right Heart` and `Left Heart`; one-way *allow* blood behaviors to all the components except `Cardiovascular Control`; and *allow* and *expel* signal behaviors to and from `Cardiovascular Control` so it can adjust cardiac output. However, because the `Pulmonary` and `Systemic Circulation` do not have *expel* blood behaviors, this model would be inadequate for any situation in which the pressure in the `Pulmonary` and `Systemic Circulation` becomes a significant factor, which is true for many cardiovascular disorders.

A better model of `Pulmonary` and `Systemic Circulation` would include *expel* blood behaviors that depend on the amount of blood that is contained and on signals from `Cardiovascular Control` to constrict blood vessels. This would be further improved by having the behaviors of `Cardiovascular Control` depend on the amount of the *expel* blood behavior of `Systemic Circulation`, reflecting the behavior of the baroreceptors.

Figure 2 is a simplified version of the `Left Heart`'s behavior in our model. The behavioral states of the `Left Heart` are `systole` and `diastole`. The synchronization of these states is controlled by signals coming through the control connection Left Heart has two containers: the ventricle, which can contain blood, and the nerves, which can "contain" signals, i.e., it is where nervous signals are received.



States:
diastole, systole
Containers:
"ventricle" of blood, "nerves" of signal
Behaviors:
allow blood from pulmonary connection to aorta connection
allow signal from control connection to nerves
pump blood from pulmonary connection to ventricle during diastole
pump blood from ventricle to aorta connection during systole, amount proportional to amount-to-contract[*move* signal from control connection to nerves, message contract]
change state from diastole to systole when [*move* signal from control connection to nerves, message start-systole]
change state from systole to diastole when [duration(systole) > systole-duration-formula]

Figure 2: Behavior of Left Heart

The two *allow* behaviors represent the paths where blood and signals may move. Both of them are one-way *allows* (indicated by "from ... to ..."; two-way is indicated by "between ... and ..."). There is a *pump* blood behavior corresponding to each state. During systole the Left Heart has a *pump* behavior out of the aorta connection, i.e., blood is pushed out; a *pump* blood behavior into the heart occurs during diastole. The former *pump* behavior is dependent on the heart's contractility, whose regulation is represented here as the strength of a signal coming into the control connection. The Left Heart changes from the diastole state to the systole state when it is signalled to do so. Changing back to the diastole state occurs when systole is finished. Systole-duration-formula stands for the expression which determines how long systole lasts.

3.3. The Behavioral Relationships between a System and its Elements

The behavioral description of a system should follow from the behavior and structure of its components and substances. In our representation, this relationship is represented by the following:

- Causal patterns. Certain patterns of behavior and structure give rise to additional behaviors. The *serial allow* pattern is the simplest to understand; two *allow* behaviors in serial lead to an *allow* behavior through the whole path. For example, two pipes which are connected form a path through the two pipes. The power of the behavioral representation is that equally simple patterns give rise to *influence* and *move* behaviors. For example, for a *move* to arise, a substance must move through some path (*allow* behaviors); some "force" must direct the movement (*influence* behaviors), and the substance must move from one place to another (containers at each end or a circuit). Thus one causal pattern is a *pump* behavior from one container to another and an *allow* behavior along the same path.

- Substance knowledge indexed by causal patterns. Once a causal pattern is discovered, the attributes of the caused behavior need to be determined. This knowledge is associated with the substance that the behaviors affect. For the *pump-move* causal pattern, the rate of the movement is related to the amount of the *pump* behavior and the resistance and other attributes of the *allow* behavior.

- Composite containers. To avoid complex descriptions of behaviors, the internal structure of a behavior (such as the internal path of an *allow* behavior) can be represented as a composite container. Composite containers also form the basis for combining *expel*, *create*, and *destroy* behaviors.

The components of Left Heart are given in figure 3. The Mitral and Aortic Valve components have one-way *allow* blood behaviors, while the Left Atrium and Left Ventricle have two-way *allow* behaviors. The Left Atrium and Left Ventricle have *expel* blood behaviors which are regulated via the atrium control connection and ventricle control connection, respectively.

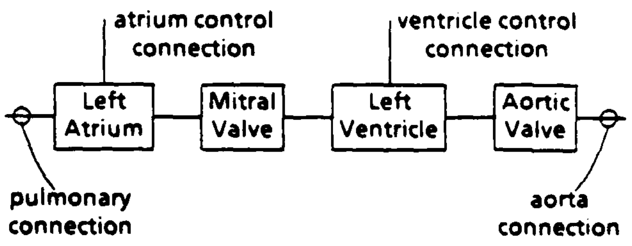
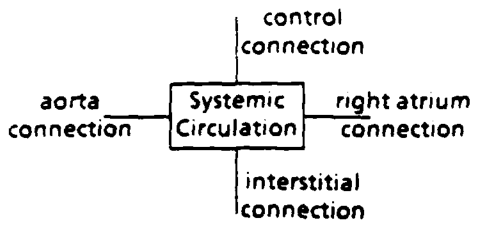


Figure 3: Components of Left Heart

The *allow* blood behavior of Left Heart (see figure 2) is caused by the *allow* blood behaviors of its components. The *pump* blood behaviors are a result of the *expel* behaviors of Left Ventricle and the *allow* blood behaviors. The *allow* signal and *change state* behaviors are taken from Left Ventricle. Figure 2 was simplified by in that it essentially ignores the *expel* blood behaviors and the states of Left Atrium. A full account of Left Heart's behavior would incorporate the *pump* behaviors caused by the Left Atrium, as well as the additional behavioral states.

3.4. Other Components of the Cardiovascular System

Figures 4 and 5 illustrate the behavior of Systemic Circulation and its structure. The *allow* blood behavior is one-way because the Veins do not allow back flow. The interaction with the interstitial connection comes from the Capillaries. The *expel* blood behavior is a combination of the *expels* of its components, primarily the Arteries and Veins. The *expel* signal behavior arises from the Arteries.



Containers:
"vessels" of blood, "nerves" of signal
Behaviors:
allow blood from aorta connection to right atrium connection, resistance proportional to amount-to-constrict[*move* signal from control connection to nerves, message constrict]
allow "blood components" between vessels and interstitial connection
allow signal between nerves and control connection
expel blood from vessels, amount proportional to amount(vessels) and constrict signal
expel signal from nerves, message pressure, amount-of-pressure proportional to amount[expel blood from vessels]

Figure 4: Behavior of `Systemic Circulation`

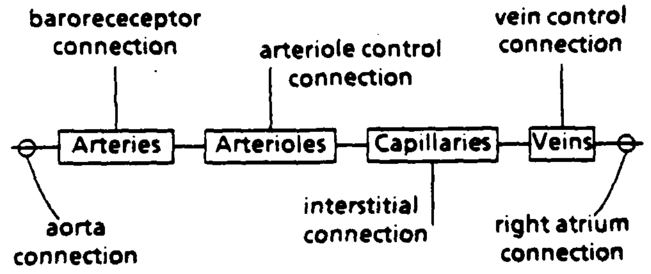


Figure 5: Structure of `Systemic Circulation`

The behaviors of `Cardiovascular Control` use the signal from the `Arteries` to send signals which regulate the behaviors of the other components. For example, the signal for contracting the heart can be represented as:

```
pump signal from control-center to left
   heart connection, message contract,
   amount-to-contract proportional to
   amount-of-pressure[move signal from
      systemic circulation connection to
      control-center, message pressure]
```

3.5. Deficiencies in the Representation

The representation is dependent on an artificial dichotomy between components and substances. The result is that it cannot properly represent the movement of components (such as the muscle and skeleton of the human body), or the connections within a substance. The reason for the problem is that the position of a component is specified by what it is connected to, and the position of a substance is indicated by what container it is in. One remedy is to develop a three-dimensional language for expressing the structure and movement of objects.

Also, the representation of substances is too limited. It does not include a notion of general classes of substances (e.g., fluids, solids) or a notion of mixtures. The difficulty is not in representing the fact that a type-subtype or a mixture relationship exists, but in understanding how such a relationship should be used to inherit or integrate knowledge about substances.

4. Example: Modeling Some Aspects of Hypovolemia

When there is a significant loss of blood, the cardiovascular system compensates in a number of ways. Some of these are directly represented in the representation, e.g., the effects of signals from `Cardiovascular Control`, while others require simulation knowledge, e.g., the distribution of the blood.

A hypovolemic condition would result in the following propagation of effects in the representation (see figure 6). First, the *expel* behaviors of the circulation components are affected since they depend on the amount of blood. The `Systemic Circulation` sends this information to `Cardiovascular Control` by its *expel* signal behavior and a *move* signal behavior between the two components. `Cardiovascular Control` then sends signals that result in (among other things) increasing the heart's contractility, and increasing the resistance and pressure of the circulation. These actions maintain and increase (if possible) the blood pressure (amount of `Systemic Circulation's` *expel* blood behavior.

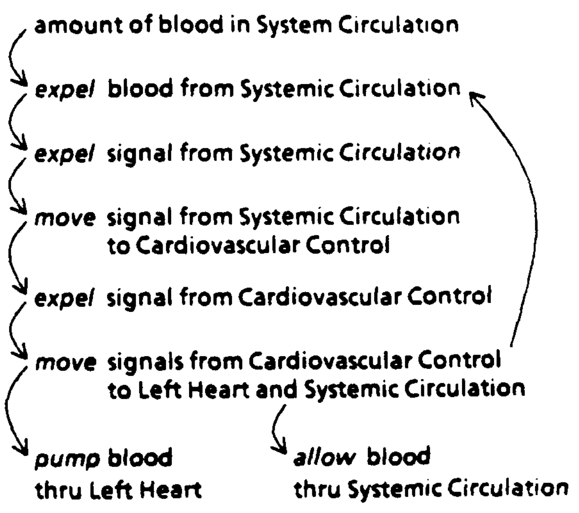

amount of blood in System Circulation
expel blood from Systemic Circulation
expel signal from Systemic Circulation
move signal from Systemic Circulation to Cardiovascular Control
expel signal from Cardiovascular Control
move signals from Cardiovascular Control to Left Heart and Systemic Circulation
pump blood thru Left Heart
allow blood thru Systemic Circulation

Figure 6: Effects of Hypovolemia in the Model

The increase in pressure is best understood by considering the components of `Systemic Circulation`, and what a simulation process would show. The pressure in the `Arteries` is directly related to the amount of blood in it. The increased activity of `Left Heart` moves more blood into the `Arteries`: the increased resistance of the `Arterioles` and the `Veins` tends to keep more blood in the `Arteries`.

Other conditions that can be partially modeled with this representation include heart congestion and fetal circulation. If cardiac failure is represented as decreased contractility, then the heart's *pump* blood behavior decreases, which results in less blood pressure. Compensation by increased venous pressure raises the blood pressure in the `Pulmonary Circulation`. If we represented the fluid flow between the lungs and the `Pulmonary Circulation`, increased flow into the lungs (which can lead to pulmonary edema) would be predicted.

To represent fetal circulation (and some heart defects), additional paths for blood can be added to the representation. Consolidation can be used to determine the new behaviors of the cardiovascular components, and propagation and simulation can report the changed blood distribution and movement.

5. Relationship to Previous Work

Our research incorporates many ideas from research on qualitative simulation. The idea of using connections and components for representing structure was borrowed from de Kleer and Brown [2]. The notion of containment was borrowed from Hayes [4] and Forbus [3]. The contribution of our work is that our behavioral representation allows for the composition of behaviors, and thus provides a basis for forming hierarchical representations.

A different way to model physical systems is through the use of causal networks. CASNET [11], ABEL [7], CADUCEUS [8], and Long's heart failure program [6] are notable examples of systems that apply causal networks to the medical domain. The primary disadvantage of causal networks is the structural and behavioral assumptions that underlie the causal links. When these assumptions don't apply (such as in septal heart defects), some other form of knowledge is needed to generate new causal links. This problem also lessens the explanatory capability of causal networks, i.e., they intrinsically cannot explain the structural and behavioral reasons behind the causal links.

6. Integrating Compiled Knowledge with Qualitative Models

One of our long-term goals is to integrate qualitative models with compiled-knowledge-based systems like MYCIN. Our current research program is to postpone working on this issue until we have developed representations and processes which satisfy the criteria listed in sections 1 and 2. However, some problems of integration can be identified.

Consider the following piece of compiled knowledge: "if low blood pressure, consider hypovolemia." One requirement is that items like "low blood pressure" and "hypovolemia" must have a corresponding translation in the model. In this case, "low blood pressure" means the amount of the `Arteries` *expel* blood behavior is below some value, and "hypovolemia" means the amount of blood in the whole circulatory system is low. Another requirement is that the model must be able to connect the two items in an efficient fashion. This is difficult to satisfy because of the potential size of the search. It may be necessary to attach a "hint" to the compiled knowledge which specifies, e.g., what other components are involved. Other representations and reasoning processes may also be needed so that inferences like "increased pulse will lead to increased output" can be quickly made. Representations such as causal networks and the functional representation of Semabugmoorthy and Chandrasekaran [9] are two possible ways to do this.

We believe that the representational framework outlined in this paper is a step toward automating reasoning about behavior. Using the cardiovascular system as an example, we have illustrated how a hierarchical representation of structure and behavior can be formulated, and how it can be used in qualitative reasoning.

Acknowledgments

We thank B. Chandrasekaran for support and criticism. This research was supported by AFOSR Grant 82-0255 from the Air Force Office of Scientific Research, grant MCS 8305032 from the National Science Foundation, and NIH Grant R01 LM 04298 from the National Library of Medicine. Dr. Jack W. Smith, Jr. is also supported by Career Development Award K04 LM00003 from the National Library of Medicine.

References

1. T. Bylander and B. Chandrasekaran. Understanding Behavior Using Consolidation. Proc. Ninth International Joint Conference on Artificial Intelligence, IJCAI, Los Angeles, 1985, pp. 450-454.
2. J. de Kleer and J. S. Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence 24* (1984), 7-83.
3. K. D. Forbus. Qualitative Process Theory. *Artificial Intelligence 24* (1984), 85-168.
4. P. J. Hayes. The Naive Physics Manifesto. In D. Michie, Ed., *Expert Systems in the Microelectronic Age*, Edinburgh University Press, Edinburgh, 1979, pp. 242-270.
5. B. Kuipers. Commonsense Reasoning about Causality: Deriving Behavior from Structure. *Artificial Intelligence 24* (1984), 169-203.
6. W. J. Long. Reasoning about State from Causation and Time in a Medical Domain. Proc. National Conference on Artificial Intelligence, AAAI, Washington, D. C., 1983, pp. 251-254.
7. R. S. Patil, P. Szolovits, and W. B. Schwartz. Modeling Knowledge of the Patient in Acid-Base and Electrolyte Disorders. In P. Szolovits, Ed., *Artificial Intelligence in Medicine*, Westview Press, Boulder, Colorado, 1982, pp. 191-226.
8. H. E. Pople, Jr. Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics. In P. Szolovits, Ed., *Artificial Intelligence in Medicine*, Westview Press, Boulder, Colorado, 1982, pp. 119-190.
9. V. Sembugamoorthy and B. Chandrasekaran. Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems. AI Group, Dept. of Computer and Information Science, The Ohio State University, 1984.
10. E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN*. Elsevier, New York, 1976.
11. S. M. Weiss, C. A. Kulikowski, S. Amarel, and A. Safir. A Model-Based Method for Computer-Aided Medical Decision-Making. *Artificial Intelligence 11*, 1 (1978), 145-172.

87-TB-CAUSAL

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
March 1987

SOME CAUSAL MODELS ARE DEEPER THAN OTHERS

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Appears in *Proc. Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, July 1987.

Some Causal Models are Deeper than Others*

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
Phone - (614)292-5835
Arpanet - bylander%osu-20@ohio-state

Submitted as a Poster to the
Ninth Annual Conference of the Cognitive Science Society

Keywords: Qualitative Reasoning, Naive Physics, Causal Reasoning

February 13, 1987

Abstract

The effort within AI to improve the robustness of expert systems has led to increasing interest in "deep" reasoning, which is representing and reasoning about the knowledge that underlies the compiled knowledge of expert systems. One view is that deep reasoning is the same as causal reasoning. Our aim in this paper is to show that this view is naive, specifically that certain kinds of causal models omit information that is crucial to understanding the causality within a physical situation. Our conclusion is that "deepness" is relative to the phenomena of interest, i.e. whether the representation describes the properties and relationships that mediate interactions among the phenomena and whether the reasoning processes take this information into account.

*This research is supported by Air Force Office of Scientific Research grant AFOSR-82-0255, and grant NIHR GOAOE 82048-02 from the National Institute of Handicapped Research.

Some Causal Models are Deeper than Others*

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University

Abstract

The effort within AI to improve the robustness of expert systems has led to increasing interest in "deep" reasoning, which is representing and reasoning about the knowledge that underlies the compiled knowledge of expert systems. One view is that deep reasoning is the same as causal reasoning. Our aim in this paper is to show that this view is naive, specifically that certain kinds of causal models omit information that is crucial to understanding the causality within a physical situation. Our conclusion is that "deepness" is relative to the phenomena of interest, i.e. whether the representation describes the properties and relationships that mediate interactions among the phenomena and whether the reasoning processes take this information into account.

1. Introduction

Most expert systems depend upon *compiled* representations and reasoning processes. Their representations associate data with conclusions, and their reasoning processes use these associations, but they do not take into account the reasons why the data and conclusions are related. Without this extra knowledge, expert systems will be limited in what explanations they can provide and in reasoning about their own limitations.**

Within AI, there has been increasing interest in *deep* reasoning, i.e. representing and reasoning about these "reasons." A number of suggestions have been made that identify deep reasoning with *causal* reasoning. Hart suggests that deep reasoning involves commonsense ideas about causality as well as mathematical modeling (Hart, 1982). Michie suggests that the fundamental laws of the domain constitute deep reasoning (Michie, 1982). A number of programs could be said to perform deep reasoning based on these criteria. Instead of summarizing and comparing these programs, which would probably be confusing rather than enlightening given the plethora of domains and reasoning methods, my strategy is to take one program and compare an explanation of its domain by the program's builders with an explanation produced by the program. The goal of the comparison is to gain insight on the relationship between "causal reasoning" and "deep reasoning."

*This research is supported by Air Force Office of Scientific Research grant AFOSR-82-0255, and grant NIHR GOAOE 82048-02 from the National Institute of Handicapped Research.

**This is not a claim that expert systems cannot perform interesting problem solving. Chandrasekaran and Mittal (Chandrasekaran, 1983) have pointed out how an expert system, for a particular reasoning situation, can fully incorporate the appropriate deep knowledge. However, it would not incorporate the deep knowledge for those situations that were not considered in its design.

2. Two Causal Explanations

These two explanations are taken from a paper by Patil, Szolovits, and Schwartz, which describes a program called ABEL (Patil, 1981), one the first programs to perform interesting causal reasoning. The first explanation is by the authors; the second by the ABEL program. The reader is forewarned that these explanations, although they concern the same domain, do not involve exactly the same phenomena.

Explanation #1. "... let us consider the electrolyte and acid-base disturbances that occur with diarrhea, which is the excessive loss of lower gastrointestinal fluid (lower GI loss). The composition of the lower gastrointestinal fluid and plasma fluid are as follows. In comparison with plasma fluid, the lower GI fluid is rich in bicarbonate (HCO_3) and potassium (K) and is deficient in sodium (Na) and chloride (Cl)... The loss of lower GI fluid would result in the loss of corresponding quantities of its constituents (in proportion to the total quantity of fluid loss)... Therefore, an excessive loss of lower GI fluid without proper replacement of fluid and electrolytes would result in a net reduction in the total quantity of fluid in the extracellular compartment (hypovolemia). Because the concentration of K and HCO_3 in lower GI fluid is higher than that in plasma fluid, there is a corresponding reduction in the concentration of K (hypokalemia) and HCO_3 (hypobicarbonatemia) in the extracellular fluid. Finally, as the concentration of Cl and Na in the low GI fluid is lower than that in plasma fluid, there is an increase in the concentration of Cl (hyperchloremia) and Na (hypernatremia) in the extracellular fluid." (Patil, 1981 - p. 841)

Explanation #2. "Moderate lower GI loss, reduced renal HCO_3 threshold, and normal HCO_3 buffer binding jointly cause no HCO_3 change. The no HCO_3 change causes low extracellular fluid HCO_3, which causes low serum HCO_3. The low serum HCO_3 and low serum pCO_2 jointly cause low serum pH. The low serum pH causes K shift out of cells and causes increased respiration rate. The increased respiration rate causes low serum pCO_2, which causes normal HCO_3 buffer binding. The low serum pCO_2 also causes reduced renal HCO_3 threshold and increased respiration rate causes increased ventilation. The lower GI loss and K shift out of cells jointly cause K loss. The K loss causes low extracellular fluid K, which causes low serum K." (Patil, 1981 - p. 898)

Both of these explanations have a causal story to tell, but in different ways and in different terms. The crucial difference is that the first quote makes use of our *physical* understanding about how the world works. It evokes a physical representation of the body and appeals to our understanding of how physical phenomena behave. The second quote is a different type of a physical explanation. While the second quote causally relates physical states, it does not express any physical relationships that let us understand the causal assertions in terms of some physical principle. Assertions like "low serum pH causes K shift out of cells" im-

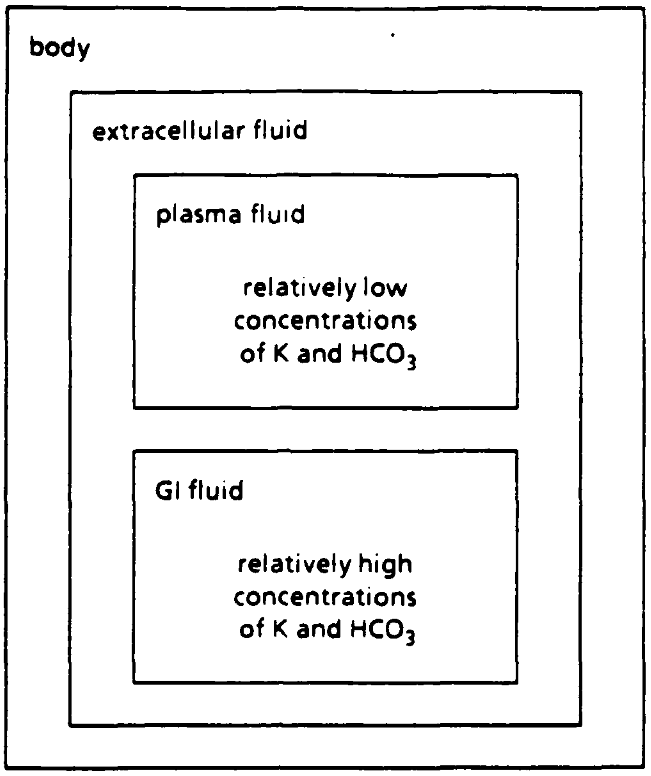

body
extracellular fluid
plasma fluid
relatively low concentrations of K and HCO_3
GI fluid
relatively high concentrations of K and HCO_3

Figure 1: Representation of Patil, Szolovits, and Schwartz's Explanation

plicitly depend on the structure of the human body and how certain parts of the body behave. ***With respect to physical phenomena, the first explanation is deep and the second explanation is compiled.***

3. An Analysis of the First Explanation

The first quote builds up the representation displayed in figure 1. (Na and Cl have been omitted for the purposes of this discussion.) The body can be thought of as having a container of extracellular fluid. The extracellular fluid compartment can be decomposed into a plasma fluid compartment and lower GI fluid compartment. Lower GI fluid has certain concentrations of HCO_3 and K, which happen to be greater than in plasma fluid. When the amount of lower GI fluid decreases (as happens in diarrhea), a corresponding amount of HCO_3 and K also decrease. It can be inferred that the total concentration of HCO_3 and K in extracellular fluid also decreases.

This representation lists the parts of the situation: fluid compartments, fluids, HCO_3, and K. It incorporates structural relationships between the parts, e.g., container, composed-of, and concentration, as well as behavioral information about them, e.g., fluid is something that can be contained, and can move. Also a fluid can be composed of other things, including HCO_3 and K in this case. The physical principle that this explanation appeals to is that when a certain amount of fluid moves, the fluid also takes what it is composed of along with it. With a little bit of qualitative (or quantitative) analysis about concentrations, it is not hard to determine how certain concentrations will increase or decrease depending on how fluid moves.

In general, reasoning about physical situations faces two problems: (1) changes in physical structure can change the overall behavior and properties of a situation. and (2) changes in a part's behavior can change the overall behavior and properties of a situation. So to perform deep reasoning about physical phenomena, representations need to express the structure and behavior of physical situations and their constituents, and reasoning processes need to be able to take this information into account. Much of the work in naive physics is aimed at reasoning about physical information such as behavioral properties of components, connections between components. and containment of substances (Hayes, 1985, deKleer, 1984, Forbus, 1984, Bylander, 1985). There has also been research on reasoning about how shape affects behavior (Forbus, 1983, Stanfill, 1983, Shoham, 1985).

4. An Analysis of the Second Explanation

The second quote is a description of the causal network illustrated in figure 2. The physical relationships that supports the causal network is not present in this explanation. For example, one part of the causal network is that loss of GI fluid contributes to low concentration of K in the extracellular fluid. However, this representation does not have structural and behavioral information such as "Extracellular fluid can be decomposed into plasma fluid and GI fluid."

Why is this additional information important? If the program only has causal networks such as in figure 2, the omitted physical information becomes a large set of assumptions that are implicitly encoded into the causal network. The result is that the robustness of the causal network depends on the likelihood that these physical assumptions are true.

For example, suppose that GI fluid in a particular person had a lower concentration of K than plasma fluid, then the causal network would be wrong. Since the causal network does not express where GI fluid sits in the body's structure and that GI fluid normally has a greater concentration of K than plasma fluid, the possibility that this information is wrong cannot be hypothesized and cannot be reasoned about. These are the same characteristics of compiled reasoning that typical expert systems have. Causal networks represent more information about associations between data and conclusions, but because they do not represent physical relationships, causal networks and their reasoning processes are also compiled.*

*Each causal link in ABEL has a "slot" for stating its assumptions. It is unclear what kind of information was being represented by the assumptions, and what reasoning processes could be performed on them. It is conceivable that a causal network could point to the information that supports it, but this additional information would be something different than causal networks.

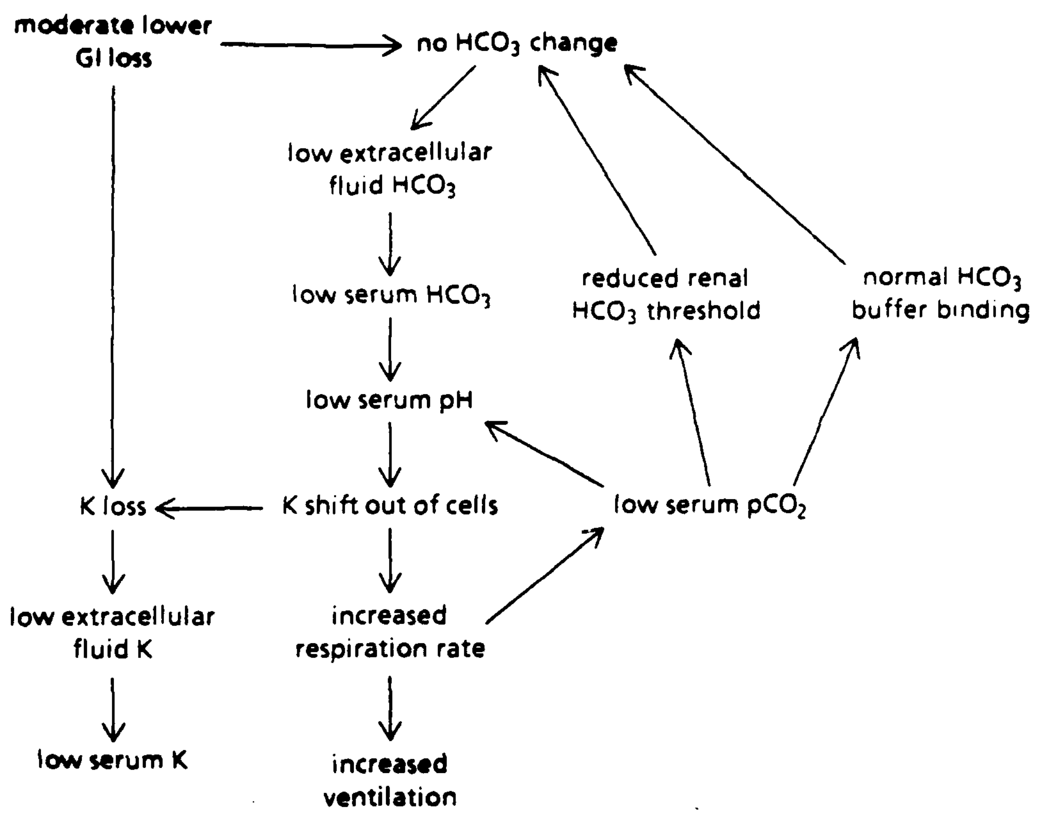


Figure 2: Representation of ABEL's Explanation

5. Some Misconceptions about Deep Reasoning

It might be claimed that representations like figure 1 are no better off than those like figure 2 because the information in figure 1 is a very qualitative representation, while figure 2 could relate physical states in more detail. This leads to the misconception that reasoning at a greater level of detail is "deeper" reasoning. This simply misses the point. Any representation worth considering can describe things at various levels of detail, but without representing physical relationships, certain kinds of reasoning processes can never be applied, no matter the level of detail.

Another misconception is that quantitative reasoning, such as solving or simulating differential equations, is deeper than qualitative reasoning. This is a misconception about the role of quantitative reasoning in reasoning about the world. A quantitative model is used when a situation can be mapped into it, and the results of applying the quantitative process can be interpreted in terms of the situation. To do this, there needs to be an understanding of what the situation is like, when the mapping is applicable, how to apply the mapping, and how to interpret the results. Each of these steps involve representation and reasoning (presumably qualitative) over and above the quantitative model. Quantitative reasoning supplements other reasoning processes; it does not substitute for them.

6. The General Nature of Deep Reasoning

On the basis of these examples, I propose the following definition of "deep":

> A representation is "deep" with respect to a class of phenomena iff the representation describes the properties and relationships by which the phenomena interact.
>
> A reasoning strategy is "deep" with respect to a class of phenomena iff the strategy reasons based on how the phenomena interact.

Relative to a certain class of phenomena, deep representations describe the properties and relationships that leads to interaction among these phenomena, and deep reasoning processes operate on this information. Because physical phenomena interact on the basis of physical structure and behavior, there need to be representational primitives whose meaning are structural and behavioral, and reasoning processes that can take this information into account.

References

T. Bylander and B. Chandrasekaran. (1985). Understanding Behavior Using Consolidation. *Proc. Ninth International Joint Conference on Artificial Intelligence*. Los Angeles.

B. Chandrasekaran and S. Mittal. (1983). Deep versus Compiled Knowledge Approaches to Diagnostic Problem-Solving. *International Journal of Man-Machine Studies, 19*, 425-436.

J. de Kleer and J. S. Brown. (1984). A Qualitative Physics Based on Confluences. *Artificial Intelligence, 24*, 7-83.

K. D. Forbus. (1983). Qualitative Reasoning about Space and Motion. In D. Gentner and A. Stevens (Eds.), *Mental Models*. Hillsdale, New Jersey: Lawrence Erlbaum.

K. D. Forbus. (1984). Qualitative Process Theory. *Artificial Intelligence, 24*, 85-168.

P. E. Hart. (1982). Directions for AI in the Eighties. *SIGART, 79*, 11-15.

P. J. Hayes. (1985). The Second Naive Physics Manifesto. In J. Hobbs and R. Moore (Eds.), *Formal Theories of the Commonsense World.* Norwood, New Jersey: Ablex.

D. Michie. (1982). High-road and Low-road Programs. *AI Magazine, 3*(1), 21-22.

R. S. Patil, P. Szolovits, and W. B. Schwartz. (1981). Causal Understanding of Patient Illness in Medical Diagnosis. *Proc. Seventh International Joint Conference on Artificial Intelligence.* Vancouver.

Y. Shoham. (1985). Naive Kinematics: One Aspect of Shape. *Proc. Ninth International Joint Conference on Artificial Intelligence.* Los Angeles.

C. Stanfill. (1983). The Decomposition of a Large Domain: Reasoning about Machines. *Proc. National Conference on Artifical Intelligence.* Washington, D.C..

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
March 1987

USING CONSOLIDATION FOR REASONING ABOUT DEVICES

Tom Bylander
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Using Consolidation for Reasoning about Devices

Tom Bylander
Laboratory for Artificial Intelligence Research
The Ohio State University
Columbus, Ohio 43210

13 March 1987

Submitted to *Artificial Intelligence*

Abstract

Given a collection of components connected in a certain way, how can the behavioral descriptions of the components be *composed* into a behavioral description of the collection as a whole? This question points out a need to seek alternatives to qualitative simulation, which has been the most common approach to generate device behavior from component descriptions. We answer this question by proposing a reasoning process called consolidation in which (1) the behavior of components are represented using a small number of primitive types of behaviors and (2) behavior is inferred using rules of composition that describe how one type of behavior can arise from a structural combination of other types of behavior. This paper discusses the basic consolidation process, reasoning processes that underlie consolidation, and the limitations of our current approach.

Acknowledgments

This research is supported by Air Force Office of Scientific Research grant 82-0255 and grant NIHR GOAOE 82048-02 from the National Institute of Handicapped Research. The research that was performed in collaboration with Jack W. Smith, Jr. and John R. Svirbely was supported by NIH Grant R01 LM 04298 from the National Library of Medicine. Jack W. Smith. Jr. is also supported by Career Development Award K04 LM00003 from the National Library of Medicine. Computer facilities were enhanced by gifts from Xerox Corporation. Thanks go to B. Chandrasekaran and Mike Tanner for their comments on an earlier draft.

Table of Contents

List of Figures

Using Consolidation for Reasoning about Devices

Tom Bylander

1. Introduction

Naive Physics is the subject of how the physical world can be understood by naive intelligent agents, who are naive because they are not students of Physics, but are intelligent because they can still reason effectively about the physical world. Because people are prime examples of such agents, and because computers have the potential for powerful reasoning, research on Naive Physics attempts to answer the questions: How do people reason about physical phenomena? How can computers be endowed with similar facilities? Artificial Intelligence research on Naive Physics concentrates on the second question, and by doing so, also seeks to achieve significant insight on the first.

In this paper, we address the following problem: Given the structure of a device, how can the behavioral descriptions of its components be *composed* into a behavioral description of the device as a whole? We propose a representation of behavior and a reasoning process, called *consolidation*, that uses the representation to draw conclusions about the behavior of subsystems of the device as well as its overall behavior. To better understand what consolidation is good for, we shall begin by contrasting consolidation with the more familiar process of qualitative simulation.

1.1. Qualitative Simulation

Much of the research effort on Naive Physics has been devoted to qualitative simulation, which is also a process for drawing conclusions about the behavior of devices [8, 14, 19, 37, 20]. The behavior of the device's parts is primarily represented via constraints on "quantities,"[1] which are qualitatively-specified real-valued variables and

[1]In Kuipers's proposal, the constraints apply to the whole device, not to individual components.

derivatives, i.e., the value of a quantity is represented as being in a real interval or at a point between two real intervals. The simulation process uses some form of constraint satisfaction to determine the possible values of the quantities during the "current" period of time and differential perturbation to update quantities for the next time period. The result of qualitative simulation is a temporal sequence of events that the device goes through. In general, the qualitative nature of the representation leads to ambiguities, so that qualitative simulation might only determine what sequences of events are possible, not which one actually occurs. [7]

There is a significant difference between the behavior input to qualitative simulation and the behavior that is output. The initial behavioral description of the device does not specify sequences of events, but expresses a collection of arithmetic relationships between quantities.[2] The problem is that the relationships which apply to a specific part severely underconstrain what sequences of events the part can go through. Instead, a reasoning process (qualitative simulation) is needed to determine the sequence of events based on the behavioral descriptions of all the parts. To distinguish these two types of behavior, we will call the behavior input by qualitative simulation *potential behavior* and the behavior that is output *actual behavior*. The idea underlying this distinction is that actual behavior expresses what events happen in what order, while potential behavior expresses behavioral characteristics independent of initial conditions and outside interactions. For example, the actual behavior of a component is the temporal sequence of values of its quantities in a particular situation, while its potential behavior is its behavioral characteristics independent of the device it is part of, or the components it is connected to, cf. de Kleer and Brown's no-function-in-structure principle [8]. Note that the concepts of actual and potential behavior are just as applicable to devices and their subsystems as to their components.

[2] De Kleer and Brown call these relationships "laws" and Forbus calls them "processes."

This distinction is important because qualitative simulation needs two other kinds of information about a device in addition to the potential behavior of its parts: the initial state of the device and the interactions between the device and the outside world. Without this information, the value of each quantity that can be affected becomes indeterminable. This is because devices, in general, can have many initial states. For example, each container within a device could be empty, full, or somewhere inbetween. Also, the world can interact with the device in many ways. Enumeration of all conceivable outside interactions is not very practical since the number, kind, and order of interactions can vary greatly

Consequently, qualitative simulation is restricted to problems of a particular type, namely those of the form:

```
input: structure of the device +
       potential behavior of its parts +
       initial state of the device +
       interactions with the outside world

output: actual behavior of the device
```

The point is not that qualitative simulation is not useful. Obviously, prediction of actual behavior is a useful and necessary activity. However, prediction by qualitative simulation is difficult if the right information is not available, and in addition, actual behavior might not be the only output that is useful. This raises the question of whether other forms of reasoning can make other kinds of useful conclusions with less information.

1.2. Consolidation

In this paper, we propose a reasoning process called *consolidation* for inferring the potential behavior of a device from the structure of the device and the potential behavior of its parts. We present a representational system for describing the structure of simple devices and the potential behavior of their parts, and a method of symbolic reasoning that processes these descriptions to infer the potential behavior of the device.

Why infer potential behavior? It would appear that if the parts' potential behavior is known, then the combination of their behavioral descriptions serves as a description of the potential behavior of the device. However, this description of potential behavior is uninteresting because, presumably, a device is constructed to have a potential behavior that could not be achieved by any of its individual parts. Since the potential behavior of a device is not equivalent to the potential behavior of its parts, some reasoning is called for to determine the device's potential behavior. Also, a description of the device's potential behavior is likely to be more concise and efficient for further reasoning about the device. If a black-box description of the device is all that is needed, then a behavioral description of the device can substitute for the behavioral descriptions of all its parts.

The problem of inferring the potential behavior of devices is only one of many Naive Physics problems. Clearly, other kinds of information are useful to derive, e.g. actual behavior, intended purpose, designs, malfunctions, etc. Also, there is the problem of determining information about the parts given information about the device. This raises the question of how an intelligent agent can coordinate its Naive Physics problem solving activities. Most of these issues are well beyond the scope of this paper. Although this paper is limited in scope, we believe that studying specific Naive Physics problems, specifying general reasoning processes to solve those problems, and determining how to coordinate different reasoning processes will be a fruitful approach to understanding Naive Physics.

A program based on our consolidation framework has been implemented and works on a number of examples, two of which are presented in this paper. Here are some examples of reasoning (translated by us into English) that this program can achieve:

- Two batteries in series behave like a battery with a larger voltage.[3]

[3]The significance of this answer is that the program derives a behavioral description that is similar to a battery. The program does not actually determine that this description is like a battery.

- Two streams of water separated by a heat conductor behave like a heat exchanger.
- A gear mechanism allows energy to be transferred from one shaft to another.
- The chambers and valves of the heart make it behave like a pump.

Many limitations and loose ends remain in the framework we propose. For instance, some substantial changes in the representation appear to be necessary in order to fully encompass the domain of devices. To the extent that this proposal is on the right track, these difficulties represent the next set of questions that future research needs to resolve.

Note: For the remainder of this paper, we use the following conventions to simplify the discussion. "Behavior" is generally used to mean "potential behavior." The reader will be warned whenever "behavior" is being used in its generic sense. "Behavioral description" is used to mean "description of potential behavior." If X is a physical object or a class of physical objects, the phrase "description of X" means "description of X's potential behavior."

1.3. Combining Components

A major limitation on any symbolic reasoner is that the behavior of a device cannot be derived directly from its structure and the behavior of its parts. Intermediate representations need to be constructed and processed. At any given time, some part or substructure of the device is being analyzed, and the rest of the device is temporarily ignored.[4] Following Simon's claim that systems are constructed and understood as hierarchical [31], it should be possible to understand a device as a hierarchical system of subdevices. This suggests a simple divide-and-conquer strategy: first infer the behavioral descriptions of subdevices, and then to synthesize these into the behavioral description of the whole device.

[4]It would be more accurate to say "For any given computing process, some part or substructure ..." With enough processes running in parallel, every part of a device might be analyzed at the same time. However, a method for analyzing substructures of devices is still necessary.

Figure 1 illustrates how we apply this strategy, and also introduces some terminology. A model of a device is displayed in the upper left corner of the figure. The device has four components, named A, B, C, and D. Connections between components are indicated by the solid lines between the circles. Connections are not a special kind of component, but serve to represent the places where components interact and are in contact with other components. Objects like pipes and wires, then, are properly treated as components rather than connections. Open connections represent potential connections of the device, i.e., where it interacts with the outside world.

The basic processing sequence of consolidation is to select a substructure consisting of two components and to produce a behavioral description of the substructure or *composite component*. In the figure, A and B are selected for consolidation, and a behavioral description of the composite component AB is produced (middle right of the figure). Successive applications of this sequence results in an analysis of the composite component ABC (a combination of AB and C), and finally the whole device.

1.4. Representing Behavior to Facilitate Composition of Behavior

It is desirable to use a simple and straightforward mechanism for inferring the behavioral description of a composite component. As previously mentioned, qualitative simulation proposals represent behavior via constraints on quantities. It would appear, then, that consolidation would need to infer the constraints and quantities of composite components from the constraints and quantities of the subcomponents. The task is not as simple as concatenating all the constraints of the subcomponents. Instead, there is a need to directly specify how the composite component interacts with other objects. Consequently, an analysis of constraints would be required, so that all the constraints that apply to the subcomponents are reduced to a more perspicuous set of constraints that expresses the composite component's behavior. This is an open and difficult issue. For example, equation operations like substitution do not apply to the confluences of de Kleer and Brown, so that inferences like the following are incorrect.

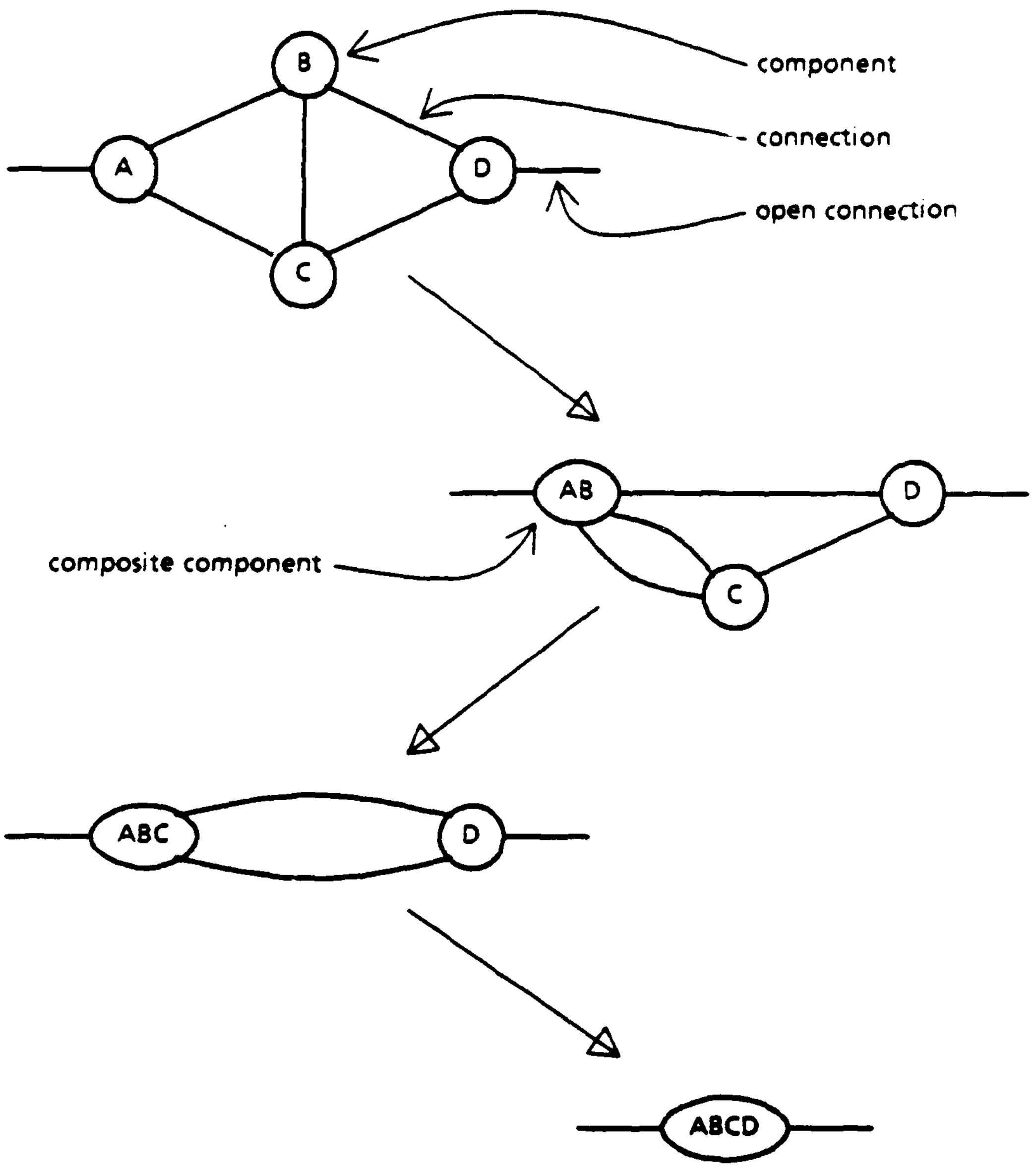


Figure 1: An Outline of the Divide-and-Conquer Strategy

$$W = Y - Z \wedge X = Y - Z \Rightarrow W = X$$

If Y is negative and Z is positive, W and X can have different signs without any contradiction in the confluence formalism, therefore $W = X$ does not follow. Difficulties like these would make it hard to do consolidation using constraints.

Instead, we propose to represent behavior based on a small number of "types of behavior." These types of behavior support rules of composition that describe how one type of behavior can arise from a structural combination of other types of behavior. We cannot demonstrate that this proposal for deriving behavior is the "simplest mechanism" that can ever be developed. However, it strongly suggests that behavioral descriptions can be inferred by straightforwardly representing and composing behavior.

The following observations illustrate how composition of behavior is possible

- Two pipes connected end to end behave like a longer pipe.
- A water pump connected end to end with a pipe behaves like a water pump.
- Pumping from one water tank to another through a pipe moves water from the first tank to the second.

It is not necessary to restrict these statements to water and components acting on water. For other types of substances, the equivalents of pipes, water pumps, and tanks can be substituted in these statements, and they still remain essentially correct.

The strategy this suggests is to represent general types of actions on substances, and to develop a set of composition rules that describe how these actions can combine to give rise to aggregate actions. The types of actions (called ***types of behavior***) and rules of composition (called ***causal patterns***) are two of the important contributions of this research. The types of behavior and causal patterns generalize the above examples in the following way:

- The *serial allow* causal pattern states that one *allow* behavior (one type of behavior is designated "allow") in series with another *allow* behavior results in another *allow* behavior over the combination of their paths. A pipe has an *allow* water behavior.

- The *propagate pump* causal pattern states that a *pump* behavior in series with an *allow* behavior results in a *pump* behavior over the combination of their paths. A water pump has a *pump* water behavior.
- The *pump move* causal pattern states that a *pump* behavior and an *allow* behavior over the same path from one container to another results in a *move* behavior over that path.

The names of behavior types and causal patterns are italicized to distinguish technical usage from conventional English. At the risk of being ambiguous, an instance of a type of behavior is often referred to as "a behavior."

The types of behavior and the causal patterns allow for a "simple and straightforward mechanism" by reducing a large part of the derivation of behavior problem to finding structural combinations of behaviors.

1.5. The Framework of Consolidation

The component composition strategy and the behavior composition strategy can be combined in the following way. The behavioral description of each component consists of the behaviors and the structural elements that are relevant to how it interacts with other components. The desired result is a behavioral description of the device that consists of the behaviors and structural elements that are relevant to how it interacts with the outside world. Note that a component (or composite component or device) can have many behaviors and many structural elements (many connections and containers). The component composition strategy can then be used to control the inference of behaviors by restricting the context (the composite component) in which inference can take place.

Besides these strategies, a number of other reasoning processes need to be invoked. Figure 2 illustrates the overall processing framework. The boxes in the figures represent in-

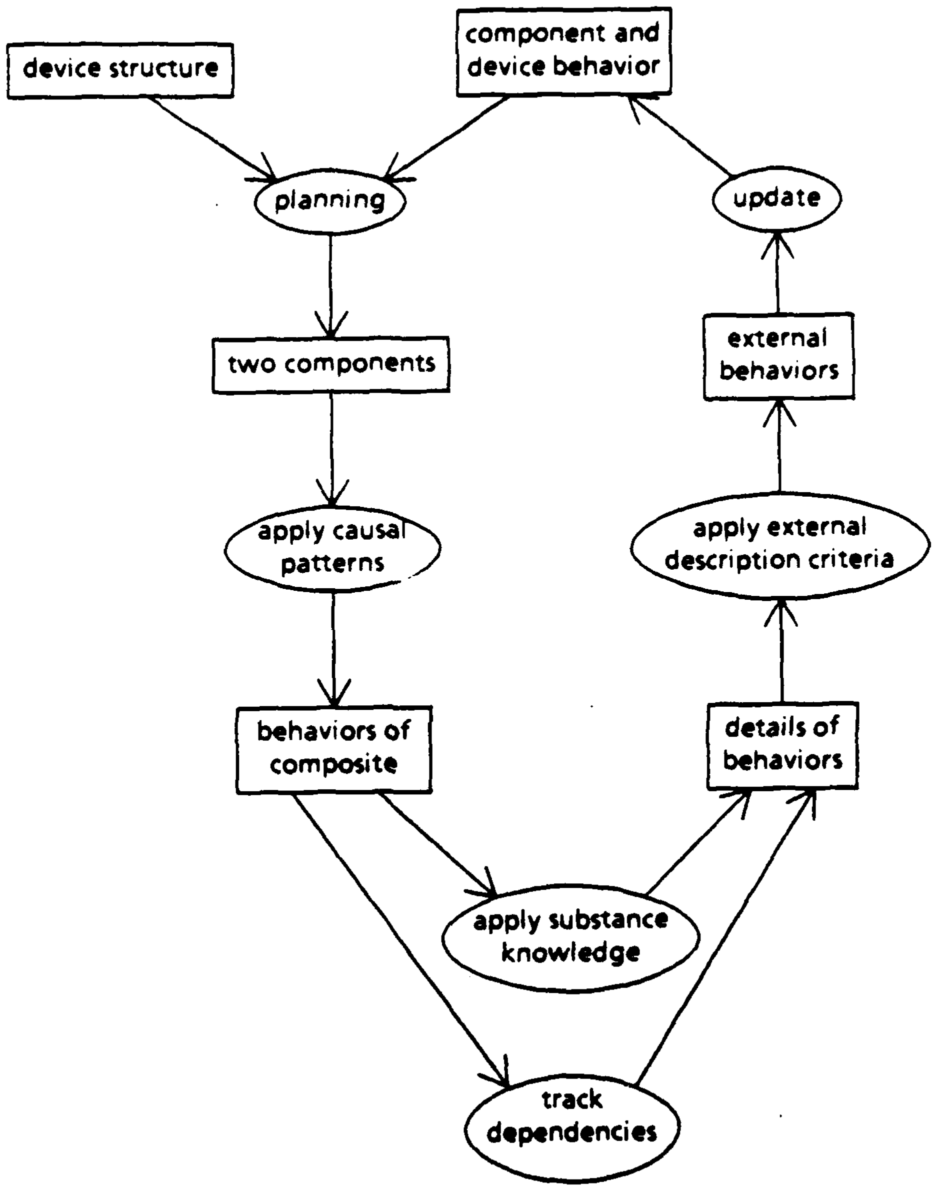


Figure 2: An Outline of the Consolidation Framework

formation that is selected or inferred and the ellipses represent processes. The input to consolidation is the structure of the device and the behavior of the components. The "Component and Device Behavior" box represents the initial information about the components' behavior plus any that is inferred. A planning process selects two components (either or both can be composite components) based on the structure of the device and what composite components have been processed so far. The causal patterns are applied to these components to determine what the behaviors of the composite component are. Further details about each inferred behavior, such as the values of its attributes, are determined by using knowledge associated with the substance being acted upon and by keeping track of any dependency relationships to other behaviors. Of the behaviors that are inferred and the original behaviors of the two components, only those that describe the composite component's black-box behavior (called its "external description") are needed for further consolidation. At this point, the behavioral description of the composite component is completed, and what is known about component and device behavior is updated accordingly.

Our research has analyzed the representations and the strategies used by each of the processes displayed in figure 2. All the representations and processes have been implemented.

1.6. Limitations of Consolidation

It is important to distinguish two types of limitations: those that apply to any consolidation process and those that are due to weaknesses and omissions in our particular framework. This section primarily explains limitations of the first kind. The main limitation, of course, is that consolidation works on only one kind of Naive Physics problem—deriving the potential behavior of a device. It does not, for example, determine the actual behavior of a device, nor does it design devices that perform some behavior.

Consolidation is limited by the availability and capability of other reasoning processes. The preciseness and succinctness of a behavioral description depends, in part, on being able to reason about the attributes of behaviors (such as amounts and rates), especially about relationships between attributes of different behaviors. This includes keeping track of ordinal relationships and reasoning about feedback.

Another limitation of consolidation is that all the behaviors of components and substances must be known. However, the precise details about any behavior are not required. For example, if a component has a *pump* water behavior, but its behavioral description does not mention it, then consolidation will likely make bad inferences, e.g., a *move* water behavior and the effects of the *move* might not be inferred. However, all the details about the *pump* water behavior are not required. A lack of detail might result in vague conclusions, but not wrong ones.

Another general problem with consolidation is that combinatorial problems can arise. The behavioral description of a composite component might have more behaviors and more structural elements than either of the subcomponents. Our framework provides for some summarization, but does not prevent a number of combinatorial problems. It is unclear whether additional summarization processes can handle all the possibilities.

The specific consolidation framework described in this paper has several weaknesses that are not necessarily inherent to all consolidation problem solvers. Our framework does not provide for spatial reasoning about shape and orientation, for certain aspects of reasoning about substances such as mixtures, for certain kinds of summarization and abstraction of behavioral descriptions, for actions at a distance such as gravity, and for focusing control on the physical phenomena of interest. These weaknesses are mentioned in passing in the sections immediately following this one, with an extended discussion of them in a later section.

1.7. Research Related to Consolidation

1.7.1. Primitives of the Physical World

In research on natural language understanding, both Schank [27, 28] and Wilks [35, 36] have proposed "primitive semantic units" (Wilks's phrase) for representing physical actions and physical objects. Many of their primitives have meanings similar to our types of behavior. For example the PROPEL action primitive proposed by Schank, which means "apply a force to," is similar to the ***pump*** type of behavior; the CONT primitive proposed by Wilks, which means "being a container," is similar to our containment structural primitive.

Clearly then, the types of behavior and the other representational primitives in our framework are not new discoveries. The key advance is that it includes a process for inferring the behavioral description of composite components and devices. Both Schank and Wilks are, of course, concerned with making inferences, but they have concentrated on the representation and processing of natural language rather than of physical situations.

Fink, Lusth, and Duran [11, 12, 22] have recently developed a set of primitives for representing the physical functions of a device's components and for simulating the device: "transformers" convert substances into other substances; "regulators" control other components; "reservoirs" store substances; "conduits" transport substances; and "joints" provide connections between components. Their primitives can be analyzed as standard combinations of the structural relations and types of behavior in our proposal. For example, "reservoir" is a combination of a container with an *allow* and an *expel* behavior. The disadvantage of their proposal is in describing the behavior of "non-standard" components and the device as a whole.

1.7.2. Hayes's Plan for Understanding Naive Physics

Much of the interest in Naive Physics is due to Hayes. who has urged the AI community to study Naive Physics and has outlined a research plan for studying Naive Physics [16, 17, 18]. The heart of Hayes's proposal is "the construction of a formalization of a sizable portion of commonsense knowledge about the everyday physical world." The first part of the plan is to represent "our own intuitive concepts" about the physical world so that the representation supports the inferences that people can make about the physical world. Constructing a program that can perform this reasoning is to be postponed until the formalization can provide the desired inferential capability.

Our contribution to the study of Naive Physics is not quite in the way that Hayes has in mind. Instead of concentrating solely on representation, we also ask the questions: What strategies can a reasoning process use to efficiently solve the problem? How can a representation support these strategies? Instead of assuming an idealized reasoning process (being able to prove or disprove anything), we also consider how the reasoning process can be done using a reasonable amount of resources. Following this strategy compels us to find representations that are useful both for describing the physical world and for reasoning effectively about it. Our desire for such use-specific representations and reasoning processes follows the spirit of recent research in knowledge-based reasoning [3, 4].

1.7.3. Causal Link Representations

Rieger and Grinberg propose a representation of physical situations that is based on describing the causal interactions (causal links) between the events that occur in a physical situation [26]. This representation is used to perform a simulation of the device. The primary components of the representation are 10 types of causal links and 4 types of events. For example, a "continuous causal" link can be used to assert that an action (a type of event) causes a state (another type of event) to occur as long as the action and other specified conditions are in effect. With a "one-shot causal" link, the action and conditions are only required momentarily.

The limitation of causal link representations is in reasoning about changes in structure and behavior. The causal links *assume* that the device will maintain its structure, and that the components will maintain their behavior. However, if any change of these types occurs, the causal links are no longer valid, and there is no way to infer the device's behavior from the new structure and behavior. Although Rieger and Grinberg's causal link representation might efficiently represent the causal associations between physical events, it does not represent how the physical world constrains the causal associations. These remarks also apply to other causal link representations [34, 23, 25, 21].

1.7.4. Reasoning about Subsystems

De Kleer has proposed a method that incorporates qualitative simulation and reasoning about subsystems of the device. His method processes the output of a qualitative simulation of a device to determine the teleology (purposeful behavior) of the device and to identify subsystems of the device and their teleology [6, 9]. However, the dependence on qualitative simulation leads to several difficulties. One is the amount of effort, a complete qualitative simulation, that is needed to determine the behavior of a subsystem. Another difficulty is that a qualitative simulation is biased by assumptions about the initial state of the device and outside interactions with the device. A third difficulty is the ambiguity of the simulation output. Because each possible temporal sequence of events produced by a qualitative simulation has a different teleological analysis, de Kleer's method must choose which sequence has the best teleology. Since the other sequences are still possible, the teleology, at best, states only how the device and its subsystems should behave, not how they truly behave.

Other research has applied constraint satisfaction techniques to subsystems independent of qualitative simulation. In particular, the work of Sussman and Steele [33] shares the notion of reducing complexity by reasoning about a group of components as a single abstract component, which is embodied in their notion of "slices." A slice is a special kind of constraint that expresses part of the combined behavior of a group of components. By applying slices it is possible to decompose a device in different ways and to derive sets of constraints that are easier to reason about. However, their proposal is unsuitable for general consolidation because slices are conditioned on groups of components rather than the constraints themselves. Thus, a slice is not a general rule about behavior, but about a particular configuration of components.

1.8. Outline of the Rest of the Paper

The following section describes the basic structures of consolidation: the types of behavior and the causal patterns. Section 3 describes other representational structures and processes that are needed to more fully describe behavior. This includes attributes of behaviors and dependency relationships between behaviors. This section also discusses simplifying behavioral descriptions for further consolidation and planning what components should be consolidated. In Section 4, we present a more complicated example of using consolidation by showing how it can be applied to a simplified model of the human cardiovascular system. Finally, we discuss the shortcomings of our consolidation framework and suggest how they can be overcome.

2. Types of Behavior

In this section, we explain how to represent potential behavior using a small number of types of behavior, and how interactions between instances of types of behavior (referred to as "behaviors") can be described by rules of composition. A type of behavior is a type of action on a substance at some location or on some path. Naturally then, there must be a language for describing structure. The rules of composition, called causal patterns, are based on the types of behavior and the language of structure. The causal patterns describe how one behavior can arise from a structural combination of other behaviors. Our discussion begins with a simple language for describing the structure of devices, and proceeds to the types of behavior and the causal patterns.

2.1. Elements of Devices

As a first approximation, the parts of a device can be separated into two classes:

- Components. These form the topology of the device. Wires, switches, pumps, and tanks are examples of components. Our use of "component" is quite general. Empty and air-filled spaces are sometimes treated as components since the behavior of a device might depend on the properties of some empty space within it.
- Substances. Components interact with other components. The interaction is not just about components, but about the "stuff" or substances that move within and between components and affect their behavior. We apply the notion of substance to a wide variety of physical phenomena. Throughout this paper, the word "substance" is used to refer to any physical phenomena that can be thought of as moving from one place to another. This is not intended to imply that Naive Physics reasoners are required to think of heat, for example, as being material, but only that heat is a type of physical thing that moves.

A device that is used to create light might have the following components: a light bulb, a switch, a battery, and wires. The kinds of substances that these components act upon are electricity, light, and the signals that make the switch turn on and off.[5]

This dichotomy between components and substances makes it difficult to represent components that also move, such as gears, shafts, and wheels. For the moment, we ignore this problem, and assume that the component/substance distinction can be strictly enforced.

[5]Signals are not strictly physical phenomena, but an abstraction of the physical level. We use signals as a convenience when the actual substance that carries the message is not important to the analysis.

2.2. The Structure of Devices

How can a component interact with other components and with substances? One part of the answer to this question is that a component has *structure*. On its exterior, it has places that are used to connect it to other components. On its interior, it has places that hold or contain substances. This suggests two types of structural relationships:

- Connection between components. A *connection* signifies that one component is attached to another component or is otherwise in meaningful spatial contact with it. An example of "meaningful spatial contact" is the relationship of the surface of a light bulb with the space around it, which in turn, might be in contact with something that interacts with light, e.g., a photoelectric cell or a prism. We will assume that each component has a fixed number of available connections to other components. The notion of connection proposed by de Kleer and Brown [8] is similar to this one. The primary difference is that they use connections to represent ideal conduits, while we prefer to use connections as structural relationships. However, our framework does not hinge on this difference.

- Containment of substances. Both components and substances might contain substances. *Containers* represent the places inside components and substances that substances can move from, move into, and be at rest. These places might not have significant capacity, so the phrase "X has a container for Y" only implies that there is some place inside X where Y can be located, not that X has a large capacity for Y. A pipe, for example, can contain water; water can contain heat and dissolved substances. We will assume that each container holds a fixed kind of substance. The importance of the notion of containment for Naive Physics theories has been pointed out by Hayes [18] and Forbus [14].

The connections of a light bulb device are illustrated in figure 3. The positive terminal of the battery is connected to end1 of the switch, end2 of the switch is connected to end1 of the light bulb. and end2 of the light bulb is connected to the battery's negative terminal. There are two *open connections*. which indicate where this device interacts with its external environment. The switch's gate connection is where an "on" or an "off" signal can be received. The surface of the light bulb is where light radiates. Strictly speaking. the wires between these components should also be represented: however. to simplify the discussion. the wires have been omitted.

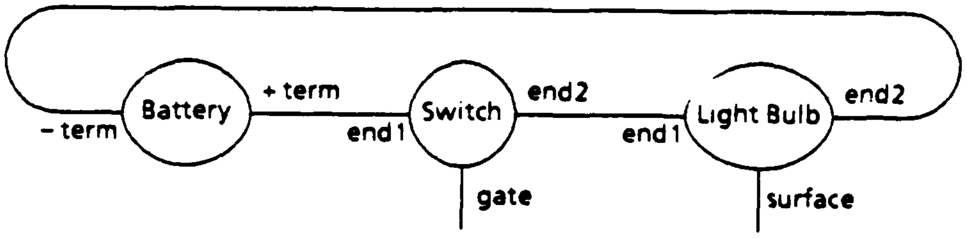


Figure 3: Connections in the Light Bulb Device

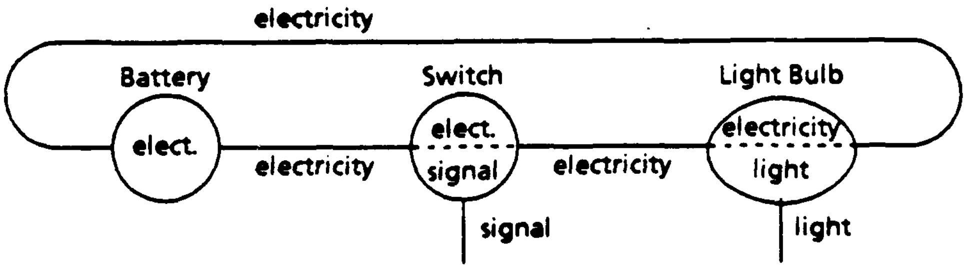


Figure 4: Substances in the Light Bulb Device

Figure 4 shows the substances that can be contained by the components of the light bulb device, as well as the substances that can move through the connections. All the components contain electricity,[6] and there are electrical connections between each components. The switch has a signal container (where it receives an "on" or "off" signal) and a signal connection. The light bulb has a light container and connection.

[6]A switch "contains" electricity because electricity can move inside a switch, not because the switch has a capacity for electricity.

Each connection, open connection, and container is treated as a location or place in the device. A "path" is a network of two or more places with two endpoints. A circuit can be described by using the same place for both endpoints. The behaviors of components describe the primitive paths from which other reasonable paths can be formed. In the light bulb device, there appears to be a circular path of electrical connections and containers through which electricity can move. At this point, however, no commitment is made about whether electricity can move or will move around the circuit. For example, when the switch is open, no electricity can flow through the switch. On the other hand, the structure does limit what paths are possible. Light, in this model, cannot move from or to the switch since the switch has no connection or container for light.

2.3. Types of Behavior

When a part of a device contains a substance, the part has the opportunity to act upon the substance, e.g., by restricting its movement, by pushing or pulling it, or by transforming it to another kind of substance. The central claim of this proposal is that a small set of primitive schemas, the types of behavior, can be used to describe these actions and reason about their interaction.

2.3.1. Allow

The simplest type of behavior is *allow*. An *allow*[7] behavior indicates that a specified substance is permitted to move on some path, i.e., from one endpoint of the path to the other. For example, a wire permits electricity to move through it, thus the wire has an *allow* electricity behavior.

Allow behaviors come in two subtypes: (1) movement is permitted in either direction, such as within a wire or pipe, and (2) movement is permitted only in one direction, such as within a diode or heart valve. These are respectively called two-way and one-way *allow*

[7]When the term "allow" is used to refer to the *allow* type of behavior, it is italicized. The names of the other types of behavior are treated similarly.

behaviors. When the details of an *allow* in one direction is significantly different from the other direction, two one-way *allow* behaviors are used.

2.3.2. Expel

Expel and *pump* are two types of behavior that describe influences or forces, i.e., an attempt to move a substance. An *expel* behavior is an attempt to move a substance from (or to) a container, e.g., a balloon has an *expel* air behavior. An *expel* behavior does not specify any path of influence, but only a single place from which an influence emanates.

Two subtypes of *expel* might be distinguished based on whether the influence is pushing the substance out of the container or pulling it into the container. However, we have chosen to represent an outward *expel* by describing the amount of the *expel* with a positive value, and an inward one with a negative value. Attributes and their values are described in more detail in the Section 3.

2.3.3. Pump

Pump is the other type of influence. A *pump* behavior is an attempt to move a substance through some path. A battery has a *pump* electricity behavior; a heart has a *pump* blood behavior; a pipe with one end higher than the other also has a *pump* behavior.[8]

There are subtypes of *pump* based on where the source or sources of the influence are, i.e., the places along the path of the *pump* where there is a "push" or a "pull." It is important to distinguish between the following subtypes:

- none of the *pump*'s sources are located at the ends of the path;
- one of the *pump*'s sources is located at one end of the path; and
- some of the *pump*'s sources are located at both ends of the path.

Their importance will become clearer when interactions between behaviors are discussed below.

[8] It would be better to say that the earth in combination with the pipe causes a *pump* behavior over the pipe. This situation, in fact, is a difficult one for this representation, but we shall postpone discussion about it for now.

2.3.4. Move

Neither the *expel* nor the *pump* type of behavior makes any commitment about whether some movement is occurring; that is accounted for by the *move* type of behavior. A *move* behavior states that a specified substance is moving from one place to another along a specified path. *Move* behaviors are constrained by the availability of a substance at one end of the path and the capability of holding or getting rid of the substance at the other end of the path. Electrical circuits often have *move* electricity behaviors. Light bulbs cause *move* light behaviors. A heat exchanger has a *move* heat behavior.

2.3.5. Create

The above four types of behavior are associated with different aspects of the movement of substances. The *create* and *destroy* types of behavior handle the appearance, disappearance, and transformation of substances. A *create* behavior states that a specified kind of substance is being created in a container. A light bulb has a *create* light behavior, as well as a *create* heat behavior. A stereo speaker has a *create* sound behavior.

2.3.6. Destroy

A *destroy* behavior states that a specified kind of substance is being destroyed in a container. An opaque material has a *destroy* light behavior. An acoustic insulator has a *destroy* sound behavior. A transformation of a substance can be represented by a combination of *create* and *destroy* behaviors.

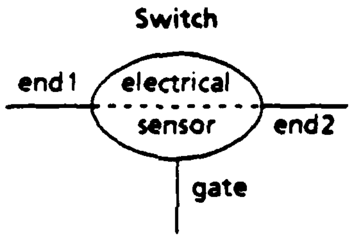
An alternative to *create* and *destroy* is a single type of behavior in which creation and destruction is specified by a positive or negative rate, respectively. This might be more appropriate for reversible phenomena.

2.3.7. Behavioral Modes and Change of Mode Behaviors

When an electrical switch is closed, it has an *allow* electricity behavior; when the switch is open, it does not have an *allow* electricity behavior. We describe the switch as having two *behavioral modes*, operating regions that are associated with different sets of behaviors. An additional type of behavior, called *change mode* specifies a predicate on behavior and the next behavioral mode of the component. For example, the switch has a *change mode* behavior from open to closed when it receives an "on" signal. A *change mode* behavior from closed to open would describe the conditions that opens the switch.

2.4. Description of the Light Bulb Device

With this repertoire of behaviors, a behavioral description of the components of the light bulb device presented in figures 3 and 4 can now be given. Figure 5 displays the behavioral description of the switch, and is an example of the format that we use throughout the rest of the paper.



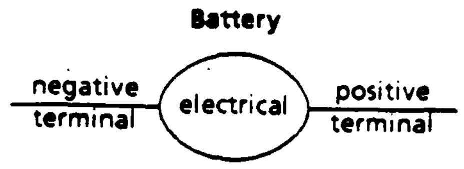
```
Connections:
    end1 of electricity
    end2 of electricity
    gate of signal
Containers:
    electrical of electricity
    sensor of signal
Modes:
    open
    closed
Behaviors:
    allow electricity between end1 and end2 thru electrical.
        mode closed
    allow signal from gate to sensor
    destroy signal in sensor
    change mode to closed
        when [move signal from gate to sensor, message on].
        mode open
    change mode to open
        when [move signal from gate to sensor, message off].
        mode closed
```

Figure 5: Behavioral Description of the Switch

The description is split into four sections. with the names of sections in boldface and keywords in italics. The first two sections describe the structure of the switch which is also pictured above the description. The connections are named **end1**. **end2**. and **gate** The containers are named **electrical** and **sensor**. The names are intended to facilitate the reader's understanding of the description. but interpreting the representation does not depend on what names are selected. Each connection and container is specific to a single substance. The next section lists the behavioral modes of the switch. named **open** and **closed**. The modes section is included in a behavioral description only if there is more than one behavioral mode.

The final section lists the behaviors of the switch. When it is in the **closed** mode. the switch has an *allow* electricity behavior from one end to the other through its **electrical** container. The phrase "*between* ... *and* ..." signifies a two-way *allow* behavior. The switch also has an *allow* signal behavior from its signal connection to its signal container. "*from* ... *to* ..." signifies a one-way *allow* behavior. The switch does not remember all the signals that are sent to it, so it must have a *destroy* signal behavior. There are two *change mode* behaviors. The switch changes its behavioral mode from **open** to **closed** when it receives an **on** signal. It changes mode from **closed** to **open** when it receives an **off** signal. **message** is a parameter associated with signals. Note that there is no section specifically for substances, but that they are specified in the declarations of connections. containers. and behaviors.

The behavioral description of the battery is given in figure 6. The battery has two electricity connections, **negative terminal** and **positive terminal**. and one electricity container. **electrical**. It has two behaviors. Its *pump* electricity behavior goes through the whole battery, and its source is the **electrical** container. The battery's *allow* behavior also goes through the battery and is a two-way *allow*.

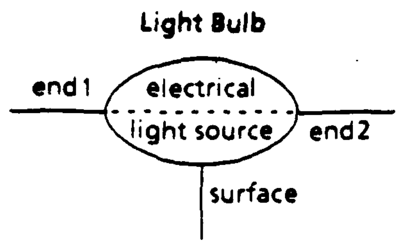


```
Connections:
    negative terminal of electricity
    positive terminal of electricity
Containers:
    electrical of electricity
Behaviors:
    pump electricity from negative terminal to positive terminal
        thru electrical, source electrical
    allow electricity between negative terminal
        and positive terminal thru electrical
```

Figure 6: Behavioral Description of the Battery

The light bulb (figure 7) has three connections and two containers. It has an *allow* electricity behavior between its two electricity connections and through its one electricity container, and an *allow* light behavior between its light connection and container. The light bulb also has *create* light, *expel* light, and *destroy* light behaviors, all of which are located in the **light source** container. The *destroy* light behavior is needed when the light that is created cannot, for some reason, move out of the light bulb. Since the light bulb is unable to store light, the light "disappears." The light is actually transformed into heat, but to simplify the example (and to avoid representational difficulties we don't want to discuss here), the light bulb's heat behaviors are omitted.

Finally, figure 8 describes the structure of the light bulb device. The names of the components are chosen for the benefit of the reader, and not the representation. Non-electrical connections in the components' descriptions, such as the surface of the light bulb, are not connected to other components, so they are not mentioned in the description.



Connections:

    end1 *of* electricity
    end2 *of* electricity
    surface *of* light

Containers:

    electrical *of* electricity
    light source *of* light

Behaviors:

    allow electricity *between* end1 *and* end2 *thru* electrical
    allow light *between* light source *and* surface
    create light *in* light source
    expel light *from* light source
    destroy light *in* light source

Figure 7: Behavioral Description of the Light Bulb

Components:

    battery *instance of* Battery
    switch *instance of* Switch
    light bulb *instance of* Light Bulb

Connections:

    positive terminal *of* battery *and* end1 *of* switch
    end2 *of* switch *and* end1 *of* light bulb
    end2 *of* light bulb *and* negative terminal *of* battery

Figure 8: Structural Description of the Light Bulb Device

2.5. Causal Patterns of Behavior and Structure

A causal pattern describes a situation in which a behavior can occur, asserting that if certain behaviors satisfy a specific structural relationship, then another behavior of a specified type could be caused.[9] For example, the *propagate pump* pattern specifies that a *pump* behavior in a serial relationship with an *allow* behavior can cause another *pump* behavior, e.g., a *pump* electricity behavior between A and B, and an *allow* electricity behavior between B and C can cause a *pump* behavior between A and C. Whether this *pump* behavior actually occurs depends on the physics of the substance and the details of the sub-behaviors. Below, we describe the causal patterns in general terms. Following subsections discuss various aspects of the causal patterns and their incorporation into the consolidation process in more detail.

The following are the causal patterns that we have identified. Besides an English description of each causal pattern, we also give a logical approximation. In the formulas, each s_i is a variable for substances, p_i for paths, and l_i for locations (connections or containers). All variables in an antecedent are universally quantified. The symbol $\overset{?}{\Rightarrow}$ indicates that the implication is conditional on other details to be taken into consideration.

- *Serial/parallel allow.* An *allow* behavior can be caused by two serial or parallel[10] *allow* behaviors.

  $allow(s_1,p_1) \wedge allow(s_1,p_2) \wedge \mathrm{serial}(p_1,p_2) \overset{?}{\Rightarrow} allow(s_1,p_3)$
  where p_3 is the serial combination of p_1 and p_2

  $allow(s_1,p_1) \wedge allow(s_1,p_2) \wedge \mathrm{parallel}(p_1,p_2) \overset{?}{\Rightarrow} allow(s_1,p_3)$
  where p_3 is the parallel combination of p_1 and p_2

  For example, two pipes with both ends connected satisfy the *parallel allow* pat-

[9] Currently, the framework does not handle situations in which the behaviors satisfying a pattern refer to different types of substances, e.g., oil and water.

[10] Roughly, two behaviors are "serial" if they share an endpoint; two behaviors are "parallel" if they have the same endpoints. A following section describes the meaning of "serial" and "parallel" in greater detail.

tern, as well as the *serial allow* pattern (the pipes form a circuit). Another condition of this pattern is that the two *allow* behaviors must permit movement in the same direction. Thus one component that permits movement from A to B and another component that permits movement from C to B, but not from B to C, would not cause an *allow* behavior from A to C.

- *Propagate expel.* A ***pump*** behavior can be caused by an *allow* behavior and an *expel* behavior that is located at an endpoint of the *allow*.

  $$\textit{allow}(s_1,p_1) \wedge \textit{expel}(s_1,l_1) \wedge \text{endpoint}(l_1,p_1) \overset{\sim}{\Rightarrow} \textbf{\textit{pump}}(s_1,p_1)$$

  For example, the *expel* air behavior of a balloon combines with an *allow* air behavior from the balloon to give rise to a ***pump*** air behavior over the same path as the *allow*. The source of the ***pump*** behavior is the "air container" of the balloon.

- ***Include expel.*** A ***pump*** behavior can be caused by a ***pump*** behavior and an ***expel*** behavior that is located at an endpoint of the ***pump***.

  $$\textbf{\textit{pump}}(s_1,p_1) \wedge \textbf{\textit{expel}}(s_1,l_1) \wedge \text{endpoint}(l_1,p_1) \overset{\sim}{\Rightarrow} \textbf{\textit{pump}}(s_1,p_1)$$

  For example, a ***pump*** air behavior into a tire (such as using an air pump to pressurize a tire) is opposed by the tire's ***expel*** air behavior. The total ***pump*** behavior is a combination of the air pump's and tire's influences. The sources of the inferred ***pump*** behavior are the sources of the ***pump*** subbehavior and the location of the ***expel*** behavior.

- ***Propagate pump.*** A ***pump*** behavior can be caused by a ***pump*** and an *allow* behavior in serial.

  $$\textbf{\textit{pump}}(s_1,p_1) \wedge \textbf{\textit{allow}}(s_1,p_2) \wedge \text{serial}(p_1,p_2) \overset{\sim}{\Rightarrow} \textbf{\textit{pump}}(s_1,p_3)$$
  where p_3 is the serial combination of p_1 and p_2

  For example, the ***pump*** electricity behavior of a battery and the *allow* electricity behavior of a wire connected to the battery results in a ***pump*** electricity behavior over the wire and battery. An additional condition is that the common endpoint of the subbehaviors cannot be a source of the ***pump*** subbehavior. In the balloon example earlier. the *pump* behavior (whose source is the balloon) cannot be propagated by another *allow* behavior from the balloon.

- *Serial parallel pump.* A *pump* behavior can be caused by two *pump* behaviors in serial or parallel.

  $pump(s_1,p_1) \wedge pump(s_1,p_2) \wedge \text{serial}(p_1,p_2) \overset{?}{\Rightarrow} pump(s_1,p_3)$
  where p_3 is the serial combination of p_1 and p_2

  $pump(s_1,p_1) \wedge pump(s_1,p_2) \wedge \text{parallel}(p_1,p_2) \overset{?}{\Rightarrow} pump(s_1,p_3)$
  where p_3 is the parallel combination of p_1 and p_2

  The *pump* electricity behaviors of two batteries in serial give rise to a *pump* electricity behavior over both batteries. The sources of the inferred behaviors is a combination of the sources of the subbehaviors. In the serial case. the shared endpoint cannot be a source for both *pumps*. In this and the previous causal pattern. conditions on the sources of *pump* behaviors are used to prevent combination's of *pumps*, *expels*, and *allows* from reusing the same sources of influence.

- *Pump move.* A *move* behavior can be caused by a *pump* behavior and an *allow* behavior, both on the same path from one place to another. or both on the same circuit.

  $pump(s_1,p_1) \wedge allow(s_1,p_1) \overset{?}{\Rightarrow} move(s_1,p_3)$

  Two containers of water connected by a horizontal pipe (which causes the *allow* behavior) result in movement if there is a *pump* behavior between the containers. A wire connecting both ends of a battery is an example of the *pump move* causal pattern over a circuit.

- *Serial/parallel move.* A *move* behavior can be caused by two serial or parallel *move* behaviors.

  $move(s_1,p_1) \wedge move(s_1,p_2) \wedge \text{serial}(p_1,p_2) \overset{?}{\Rightarrow} move(s_1,p_3)$
  where p_3 is the serial combination of p_1 and p_2

  $move(s_1,p_1) \wedge move(s_1,p_2) \wedge \text{parallel}(p_1,p_2) \overset{?}{\Rightarrow} move(s_1,p_3)$
  where p_3 is the parallel combination of p_1 and p_2

One other causal pattern pertains to situations in which one substance contains another substance.

- *Carry move.* A *move* behavior of a substance s_1 that can contain a substance s_2 (e.g. water can contain heat) can cause a *move* s_2 behavior along the same path.

  $$move(s_1, p_1) \wedge contains(s_1, s_2) \overset{?}{\Rightarrow} move(s_2, p_1)$$

  For example, when something that contains heat moves from A to B, heat also moves from A to B.

We do not claim that this list is complete. Additional patterns might be required to reason about concepts like momentum, in which movement leads to additional influences, and about forces like gravity, in which one object causes influences at a distance. The absence of any causal patterns that infer *expel*, *create*, or *destroy* behaviors is also notable. This is because no structural pattern of behaviors will lead to the inference of any of these types of behavior at a new location; the capability to *expel*, *create*, or *destroy* must have been inherently there in the first place.

Suppose that a composite component of the battery and the switch of the light bulb device (refer to figure 3) is chosen for processing. The behaviors of the battery-switch are inferred as follows:

- Using the *serial allow* pattern, an *allow* electricity behavior between the negative terminal of the battery and end2 of the switch is inferred from the *allow* electricity behavior of the battery between its terminals and the *allow* electricity of the switch between its two electricity connections.

  $$\begin{aligned}&allow(\text{electricity}, <\text{negative terminal}, \text{electrical}, \text{positive terminal}>_{battery})\\ &\wedge\ allow(\text{electricity}, <\text{end1}, \text{electrical}, \text{end2}>_{switch})\\ &\wedge\ \text{serial(battery path, switch path)}\\ &\Rightarrow allow(\text{electricity}, <\text{negative terminal}_{battery}, \ldots, \text{end2}_{switch}>)\end{aligned}$$

- Using the *propagate pump* pattern, a *pump* behavior from the negative terminal of the battery to end2 of the switch is inferred from the *pump* electricity behavior of the battery between its terminals and the *allow* electricity of the switch between its two electricity connections.

pump(electricity, <negative terminal, electrical, positive terminal $>_{battery}$)
$\wedge$ *allow*(electricity, <end1, electrical, end2$>_{switch}$)
$\wedge$ serial(battery path, switch path)
$\Rightarrow$ *pump*(electricity, <negative terminal$_{battery}$, ..., end2$_{switch}$>)

The causal patterns are similar to Forbus's individual views and process descriptions [14]. They all are for expressing the conditions that give rise to behavior. The main difference is that the causal patterns are intended to be generic inferences for all substances. Although individual views and process descriptions can be stated at a high level of generality, there is no commitment in Forbus's theory to any particular level of generality, e.g., in practice, different types of substances such as liquid, gas, and heat have different process descriptions. Within Forbus's theory, the causal patterns might be expressed as "universal" individual views and process descriptions.

2.6. Structural Relationships and Properties Used by the Causal Patterns

Recognizing a causal pattern requires finding the behaviors of the appropriate types and determining that the paths and locations of the behaviors satisfy specific structural relationships. Since the behaviors of the device's parts are directly specified, finding the behaviors is the easy part. However, in order to avoid anomalies, the serial and parallel structural relationships need to be carefully defined and other structural properties need to be introduced.

2.6.1. Serial and Parallel Relationships

The serial relationship cannot be as simple as "having one endpoint in common." For example in figure 9, *allow* behaviors from A to B through D, and from B to C through D can be inferred, but it would be improper to infer, from these two behaviors, an *allow* behavior from A to C. The reason is because both the A-B and B-C *allow* behaviors go through the B-D segment. In a device with many possible circuits, the number of *allow* behaviors would become very large if *allows* are permitted to "retrace" any part of their paths. For the serial relationship then, the two behaviors cannot have any path segments in common. From a similar argument, a stricter condition can be formulated. Except for endpoints, the two behaviors cannot have any connections or containers in common.

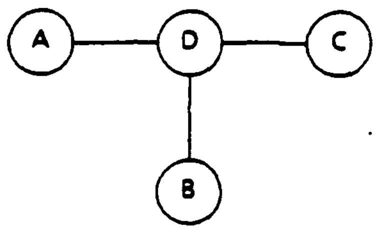


Figure 9: A Situation Illustrating a Difficulty in Defining "Serial"

For a different reason, the parallel relationship cannot be as simple as "having both endpoints in common." In figure 10, there are four different paths from A to D: A-B-D, A-C-D, A-B-C-D, and A-C-B-D. The last two paths share path segments with each other and with the first two paths, yet all four paths should be combined into a single *allow* behavior. Thus sharing path segments should not prevent *allow* behaviors from being parallel. However, it would be an obvious mistake to combine the A-B-D path with any path that already includes the A-B-D path. Thus, there is a need to prevent an *allow* behavior from being combined with *allow* behaviors that already incorporate it.

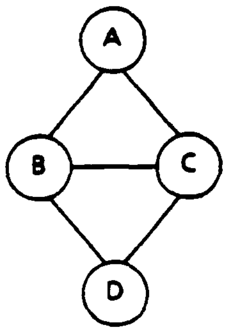


Figure 10: A Situation Illustrating Some Difficulties in Defining "Parallel"

Due to the problems with the simple definitions of "serial" and "parallel," we define them as follows:

- Two behaviors are *serial* to each other if and only if they have one endpoint in common, neither behavior is a circuit, and, except for endpoints, the paths of the behaviors do not intersect. Both endpoints are allowed to be in common, in which case the two behaviors form a circuit.
- Two behaviors are *parallel* to each other if and only if they have both endpoints in common, neither one of them is a circuit, and each contains a path segment not included in the other.

2.6.2. The Potential-End-of-Move Property

The above description of the *pump move* causal pattern would infer a *move* behavior between the terminals of the battery since the *pump* and *allow* electricity behaviors of the battery are on the same path. This inference is not useful because a *move* cannot begin or finish at either of the battery's terminals: a *move* cannot actually happen without additional paths. To avoid inferring *moves* over paths with endpoints that cannot serve as the source or sink of a *move*, we introduce a property called "potential-end-of-move," defined as follows:

- A place is a *potential-end-of-move* if and only if it is a container of significant capacity, the location of an *expel*, *create*, or *destroy* behavior, or the source of a *pump* behavior.

The definition is intended to cover the possible ways to provide or absorb a substance. If a place has significant capacity, a substance can be stored for movement either to or from the place. If a place is the location of an *expel* behavior or a source of a *pump* behavior, it is possible that a substance moving to the place could be further moved elsewhere. If a place is the location of a *create* or *destroy* behavior, then there is a way to directly provide or absorb a substance.

The *pump move* causal pattern, then, needs to be modified so that the endpoints of the subbehaviors are required to have the potential-end-of-move property. It should be noted that this property is not required for circuits.

2.6.3. The Propagatable Property

Consider the diagram in figure 11. Suppose there are *allow* behaviors for each of the path segments, and that there are *pump* behaviors from A to B (source at A), and from C to B (source at C). The *propagate pump* and *serial pump* causal patterns, as described above, would create a problem, namely that more than one *pump* behavior would be inferred on the A-B-C path. On the A-B-C path, the *propagate pump* causal pattern would combine the A-B *pump* with the B-C *allow* and the C-B *pump* with the B-A *allow*. Also the *serial allow* causal pattern would combine the A-B *pump* with the C-B *pump*.

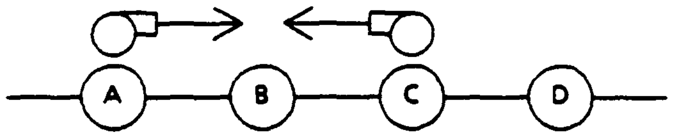


Figure 11: A Situation Illustrating a Problem in the *Propagate Pump* Causal Pattern

To avoid inferring a multiplicity of *pumps* over the same path, a property called *propagatable* is introduced to constrain the situations where the *propagate pump* causal pattern can be applied. "Propagatable" is defined as follows:

- A place is *propagatable* if and only if it is not the location of an *expel* behavior. In figure 11 all the places are propagatable.
- A path is *propagatable* if all the places within the path (all places excluding endpoints) are propagatable, and if there is no *pump* behavior with a path segment and source on the path. In the figure, C-D is propagatable, while A-B and B-C are not.
- An *allow* behavior is *propagatable* if and only if its path is propagatable.

The *propagate pump* causal pattern can then be modified so that the *allow* subbehavior and the common endpoint of the subbehaviors must be propagatable. The *propagate expel* causal pattern must be similarly modified to require the *allow* subbehavior to be propagatable. In figure 11. this would prevent the A-B and B-C *allow* behaviors (and any *allow* behaviors inferred from them) from participating in the *propagate pump* causal pattern, permitting only the *serial pump* causal pattern to reason over those portions of the path. A slight complication occurs when the *propagate expel* causal pattern is matched--the *allow* subbehavior and any *allow* behaviors inferred from it must be marked as not propagatable.

2.7. The Effect of Behavioral Mode on the Causal Patterns

The behaviors inferred for the battery-switch need to take the behavioral mode of the switch's *allow* behavior into account. Since the battery's behaviors always happen (more precisely, the model asserts that they always happen) and the switch's *allow* behavior occurs only during the closed mode, the behavioral modes of the inferred behaviors is also "closed." In general, the behavioral mode of an inferred behavior is the intersection of the behavioral modes of the subbehaviors. However, two kinds of interactions between causal patterns and behavioral modes affect the inference of behaviors and the calculation of behavioral mode.

2.7.1. Interaction with Parallel Patterns - Split Inference

Consider the situation in figure 12, in which a light bulb and a switch are parallel to each other. The ***parallel allow*** causal pattern would infer an ***allow*** electricity behavior between A and B through the switch and light bulb during the closed mode of the switch. However, during the open mode of the switch, it is important to remember that there is still an ***allow*** electricity behavior through the light bulb. In this case, the "unused mode" of the light bulb's ***allow*** electricity behavior (the ***parallel allow*** inference does not infer anything for the open mode) needs to be noted for further reasoning.[11]

[11] A similar action needs to be performed when the *parallel allow* causal pattern combines a one-way with a two-way *allow* behavior. The "unused direction" of the two-way behavior needs to be specially noted.

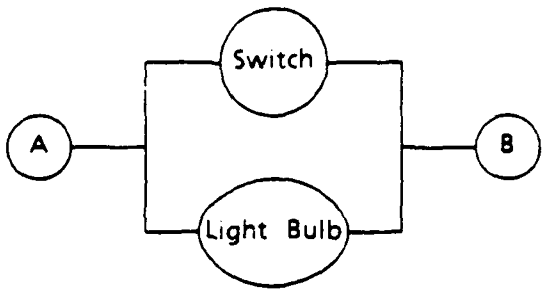


Figure 12: Switch and Light Bulb in Parallel

This type of inference, which we call *split inference*, needs to be performed for other kinds of causal patterns that involve parallel structures besides the *parallel allow* causal pattern, namely the *parallel pump*, *parallel move*, *include expel*, and *pump move* causal patterns. The latter two patterns, although they do not depend on the parallel structural relationship, infer behaviors that have the same endpoints as one of their subbehaviors, and consequently, the overall behavior between those endpoints might be divided among the inferred behavior and the unused portion of the subbehavior.

2.7.2. Interaction with Structural Properties

As defined, a place is a potential-end-of-move if, e.g., it is the location of a *create* behavior. The *create* behavior indicates that some substance is produced, and thus, movement from that location might occur. However, this is not true if the *create* behavior only occurs during a particular behavioral mode. In other words, the *create* behavior makes its location a potential-end-of-move only during the modes when it is active. A similar scenario can be given for the propagatable property as well. A place, path, or *allow* might be propagatable only during certain behavioral modes.

This complicates the causal patterns in two ways. One, the values of these properties for a place, path, or *allow* cannot be simply true or false, but must indicate the behavioral modes for which the properties are true. Two, the calculation of the behavioral mode of inferred behaviors must take this information into account in addition to the behavioral modes of the subbehaviors.

As an illustration of the consequences, this is the modified definition of the *propagate-pump* causal pattern.

- *Propagate pump.* A ***pump*** behavior can be caused by a ***pump*** and an *allow* behavior in serial. The sources of the inferred ***pump*** are the sources of the ***pump*** subbehavior. The *allow* subbehavior and the common endpoint of the subbehaviors must be propagatable. The behavioral mode of the inferred behavior is the intersection of behavioral mode of the ***pump*** subbehavior, the *allow* subbehavior, the propagatable property of the *allow* subbehavior, and the propagatable property of the common endpoint.

2.8. Behaviors of the Light Bulb Inferred with the Causal Patterns

Consider the combination of the battery-switch (refer to the end of Section 2.5) and the light bulb in the device of figure 3. The causal patterns allow the following behaviors to be inferred.

- Using the ***serial allow*** causal pattern, an *allow* electricity behavior around the circuit during the closed mode of the switch is inferred.
- Using the ***propagate pump*** causal pattern, a ***pump*** behavior around the circuit during the closed mode is inferred.
- Using the ***pump move*** causal pattern, a ***move*** behavior around the circuit during th closed mode is inferred.
- Using the ***parallel allow*** causal pattern, *allow* behaviors are inferred from the battery-switch connection (the connection between the battery and the switch) to the battery-light bulb connection (one part of the path goes through the battery and the other through the switch and the light bulb), from the battery-switch connection to the light bulb-switch connection, and from the light bulb-switch connection to the battery-light bulb connection. All the behaviors occur during the closed mode of the switch.

- Split inferences on the *parallel allow* inferences point out a number of behaviors. For each subpath that does not go through the switch, an *allow* behavior during the open mode of the switch is inferred.

3. Other Representational and Processing Considerations

Although the causal patterns provide for the commonsense inference of a significant portion of device behavior, they lack the capability to handle the following aspects of behavior

1. The causal patterns do not account for differences between substances. In the example so far, it did not matter what substance was the object of the behaviors as long as certain behaviors acted upon the same substance.
2. The causal patterns do not account for interactions between behaviors. For example, an interaction crucial to the behavior of the light bulb, but not represented, is the relationship between the flow of electricity and the production of light.
3. Finally, additional processing is needed to avoid combinatorial difficulties in consolidation. Remembering all the behaviors of the individual components as well as those inferred will present a combinatorial problem. Using the causal patterns exhaustively will result in a search of all paths in the device not to speak of parallel combinations of them.

This section discusses our solutions to some of these problems.

3.1. Representing more Detail about Substances

3.1.1. Attributes of Behaviors

Consider the *allow* electricity behavior of the light bulb:

allow `electricity` *between* `end1` *and* `end2` *thru* `electrical`

There is more to say about this *allow* behavior than simply describing its path. The path has certain characteristics that affect how electricity travels through it. In particular, this path has something called *resistance* that can be measured, i.e., that has a specific value. Characteristics like resistance that describe behavior or structure in more detail are called *attributes*.

Most of the behaviors have a natural attribute that measures its size: *move* by rate of movement, *create* by rate of creation, *destroy* by rate of destruction, and *expel* and *pump* by amount of influence. Also, some behaviors, especially *allow* behaviors, might have special attributes that have a specific meaning with regard to the substance, e.g., inductance in electricity. The attributes of a behavior should be a function of the type of behavior and the type of substance. Naturally, different theories of the world might be more or less detailed (i.e., have more or fewer attributes) depending on the amount of expertise that is desired or learned.

All the behaviors of the light bulb device have attributes. Each *allow* electricity behavior has a resistance attribute. The *pump* electricity behavior of the battery has an amount of influence (voltage). The *create* light behavior of the light bulb has a rate and a color (the latter is not represented). The *allow* signal behavior of the switch might restrict the kinds of "messages" th[illegible] pass through it.

Containers have attributes of capacity and amount. The containers of the light bulb device can be modeled with "zero" or "infinitesimal" capacity, so interesting issues concerning these attributes do not arise. Below, we will present an example in which the capacities of containers have significant behavioral consequences.

Because our interest in this paper is in showing how the behavioral reasoning is controlled and not how arithmetic reasoning is performed, we have not incorporated a framework for arithmetic reasoning such as Simmons [30]. However, we should note that consolidation requires few assumptions about the values of attributes. An attribute for a behavior or container at any single point in time has a single value of a particular type. We do not impose any restrictions on what types are allowable. The value of an attribute might be an integer, real number, vector, element of a discrete set, or whatever. For example, the resistance attribute of an *allow* electricity behavior must be a non-negative real number, while the message attribute of a *move* signal behavior might be restricted to on or off.

The issue of representing and reasoning about attributes has been addressed in some detail for the case when the values of attributes are restricted to be continuous, qualitatively-specified real numbers [10, 14, 19, 37, 38], which have been called *quantities*. However, our commonsense reasoning appears to use types of values that typically are not handled by the above work. The value of an attribute might be an integer, e.g., the number of marbles contained in a jar, or might not be a number at all, e.g., the color of a substance.[12]

3.1.2. Integrating Knowledge about Substances with the Causal Patterns

In the combination of the battery and switch, two new behaviors were inferred based on the causal patterns. However, the causal patterns do not supply the values of the behaviors' attributes, for instance, the resistance of the *allow* electricity behavior through the battery and the switch. Knowledge about electrical resistance is clearly required, but it would ruin the general nature of the causal patterns if they included specific knowledge about electricity, as well as about all other substances.

[12]Color could be an attribute of *expel* and *move* light behaviors. Of course, color is expressible as numbers, i.e. the spectrum of the light waves, but we doubt that most people reason about light that way.

Another problem is that the causal patterns sometimes infer behaviors that do not occur. For example, in figure 13 there are two batteries that oppose each other. If they have equal voltage, there is no voltage from A to B, so there is no *pump* electricity behavior from A to B. However, the *serial pump* causal pattern infers a *pump* electricity behavior whether the voltages of the two batteries are equal or not, but inserting knowledge of this type into the causal patterns would detract from their generality.

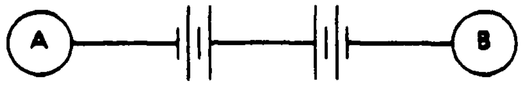


Figure 13: A Device in Which an Inferred Behavior does not Occur

The answer to these problems is to separate knowledge about substances from the causal patterns, but to organize substance knowledge around the causal patterns. That is, for each causal pattern and each substance, there can be a chunk of knowledge that indicates how to compute the attributes of the inferred behavior from the subbehaviors. For example, when the *serial allow* pattern is satisfied on two *allow* electricity behaviors, part of the "*serial allow*-electricity" knowledge says to compute the resistance of the inferred behavior by adding the resistances of the subbehaviors. Similarly, when the *propagate pump* pattern is satisfied on a *pump* and *allow* electricity behaviors, there is knowledge that asserts that the voltage of the inferred *pump* behavior is equal to the voltage of the *pump* subbehavior.

The substance knowledge can also be given the responsibility to determine whether the inferred behavior is spurious. In figure 13, the *serial pump*-electricity knowledge can determine that the voltage is zero, and on the basis of this information, can undo the inference. Thus in addition to computing the values of attributes, knowledge about substances also can determine whether the inferences made by the causal patterns are reasonable with regard to the specific situation. The role of the causal patterns is to hypothesize behaviors based on general information, and the role of the substance knowledge is to figure out the details, ruling out any behaviors that do meet certain requirements.

The claim is this: Knowledge about each substance should be organized around the kinds of behavioral inferences that can be made. This simplifies both the causal patterns and the substance knowledge. The causal patterns do not need to incorporate details about substances, and each substance does not need to have its own behavior inference mechanism.

The substance knowledge for the parallel patterns is more difficult to specify than for the serial patterns, mainly because parallel behaviors can share path segments. For some of these situations (such as in figure 10), an exact analysis would require complicated computations, such as those to solve the equations derived from Kirchoff's Laws. However, if some loss of information is acceptable, a less demanding computation can be done, e.g., the "normal" parallel computation for electrical resistance gives a lower limit on the resistance of parallel *allow* electricity behaviors.

In this paper, we do not provide a language for specifying substance knowledge, but do assume that substance knowledge is available as needed. In our implementation, a procedure was coded for each causal pattern-substance situation that occurred in the examples we used. Although this is unsatisfying as a complete theory of behavioral reasoning, it is sufficient to show how reasoning about attributes fits with consolidation. That is, the causal patterns provide the context for applying specific knowledge about substances, and thus controls reasoning about attributes and provides a basis for the organization of substance knowledge.

3.2. The External Description of a Composite Component

The battery-switch composite component not only has the behaviors that were inferred using the causal patterns, but also all the original behaviors of its subcomponents. The structure of the battery-switch is the union of the switch's structure with the battery's structure, except that the end1 connection of the switch is the same as the positive terminal of the battery. It would appear that as composites include more components, their behavioral descriptions become more complex. making it more difficult to use them in consolidation. There are several sources of complexity:

- The number of behaviors. Since consolidation is mostly based on matching behaviors. the number of behaviors of the original components and those that inferred from them potentially make consolidation more complex for larger composite components. For complex structures like figure 10. the total number of behaviors can increase rapidly. exponential in the worst case.
- The number of structural elements. Composite components, of course. have all the structural elements of their subcomponents. Inferred behaviors become more complex because they have more internal structure. For example. the inferred allow behavior of the battery-switch needs to refer to five structural elements to describe its path. Because there can be many different combinations of paths, the size of the behaviors' structural descriptions may become very large.
- The number of behavioral modes. The behavioral modes of a composite component are the cross-product of the sets of behavioral modes of its subcomponents. Since each behavioral mode might have its own corresponding set of behaviors, this potentially results in a combinatorial increase in the number of behaviors.

Is all this detail needed to describe a composite component for further consolidation? The answer is no as long as some loss of information is acceptable. That is, by summarizing or "forgetting" details, a more compact, efficient description of a composite can be derived. The drawback is that where details make a difference, such as the application of Kirchoff's Laws to circuits with structures like figure 10. some loss of information is inevitable.

Our framework provides for the summarization of a composite component in two ways.

One. *composite containers* are created as combinations of containers and connections[13] Whenever a causal pattern infers a behavior that goes through two or more structural elements. a composite container is derived from them. For example. the inferred *allow* behavior of the battery-switch would go through a composite container formed out of the electrical containers of the battery and switch. and the connection between the battery and switch. The inferred *pump* behavior goes through the same composite container as the inferred *allow*. There is one exception: if the inferred behavior is over a circuit. no composite container is made. If a composite container were made for a circuit. the direction of the behavior would be lost.

Two. only those behaviors. containers. and connections that describe the *external* behavior of the composite are selected for further consolidation. Many behaviors. containers. and connections of the composite become irrelevant for describing how the composite component behaves with respect to the rest of the device. Those that are relevant become the external description of the composite. For example, the *allow* electricity behavior of the switch is irrelevant to the external behavior of the battery-switch because the inferred *allow* electricity behavior incorporates all the useful information about the switch's *allow* behavior. i.e.. useful for further reasoning about the overall behavior of the light bulb device.

Below we propose criteria for determining the external description of a composite. These criteria are conservative in that some behaviors that are not external are selected. However, they prune much of the total description of the composite. Each criteria tests three things: one, whether the behavior operates on "external" structural elements: two. whether the behavior can be used in a causal pattern for future consolidation: and three. whether the behavior is redundant, i.e.. whether another behavior that also satisfies the other two criteria makes this behavior unnecessary to remember. Some definitions are useful for describing the criteria.

[13]The term "composite" by itself is always used to refer to a composite component. never a composite container.

A connection is an *external connection* of a composite if it connects the composite to other components or is a connection of the device to the outside world

A container is an *external container* if it is an endpoint of an external *allow* behavior.

These are the criteria:

1. *Allow* behaviors. An external *allow* behavior must satisfy one of the following:
   a. The *allow* behavior is propagatable and its endpoints are external connections or potential-end-of-moves. The *allow* behavior is redundant, however, if it has been used in a *parallel allow* causal pattern to infer an *allow* behavior that is also propagatable. This remembers *allow* behaviors for the *serial parallel allow* and *propagate pump* causal patterns. *Allow* behaviors that do not go through the composite component or do not end in a potential sink or source within the composite cannot result in inferring future *moves*.
   b. The *allow* behavior is on the same path as an external *pump* behavior. This is needed for the *pump move* causal pattern.
   c. The structure of the *allow* behavior corresponds to the difference or the intersection of an external propagatable *allow* behavior with an external *pump* behavior. This is needed to distinguish between the propagatable and non-propagatable portions of intersecting external behaviors.
2. *Pump* behaviors. One of the following criteria must be satisfied:
   a. The endpoints of the *pump* behavior are external connections or containers. However, the behavior is redundant if it has been used in an *include expel*, *pump move*, or *parallel pump* causal pattern.
   b. The *pump* behavior is on the same path as an external *allow* behavior, but has not been used in an *include expel* causal pattern.
3. *Move* behaviors. The endpoints of an external *move* behavior must be external connections or containers, and the *move* has not been used in a *parallel move* causal pattern.

4. *Expel* behaviors. An *expel* behavior is external if it is located at an external container and has not been used in a *propagate expel* or *include expel*.

5. *Create* and *destroy* behaviors. A *create* or *destroy* behavior is external if it is located at an external container.

6. *Change mode* behaviors. A *change mode* behavior is external if it involves behavioral modes that determine what external behaviors are active.

7. Any behavior inferred from external behaviors, i.e., if all the subbehaviors of a causal pattern used to infer the behavior are external.

In the battery-switch, the two inferred behaviors are external because their endpoints are external connections and they have not been used in a parallel inference. The *change mode* behaviors of the switch are also external because the open and closed modes of the switch determine whether the inferred behaviors are active. The other behaviors of the battery-switch are internal, primarily because each behavior has an endpoint that is an internal connection. Consequently, the electrical containers of the battery and the switch, and the connection between them do not need to be referenced in the external description of the battery-switch.

These two summarization features, composite containers and external description criteria, are not sufficient to handle all the problems that can occur. In particular, the number of behavioral modes can be still be very large, creating a need to compose behavioral modes in some manner. We have not determined how composition of behavioral modes can be done. Also, if a device has many containers that are potential-end-of-moves, the external description criteria ensures that they are kept in the external description. We have not resolved this issue either.

3.3. Dependencies between Behaviors

3.3.1. Representation of Dependencies

The value of an attribute can be used to express how one behavior is dependent on other behaviors. For example, the *create* light behavior of the light bulb is dependent on the movement of electricity through the light bulb. To express this *dependency*, the rate of the *create* behavior can be described as:

```
proportional (magnitude (rate
    [move electricity from end1 to end2 thru electrical])))
```

That is, the rate of light creation has a specific relationship with the rate of electricity movement through the light bulb. Whenever the rate of movement can be determined, it can be substituted in this expression, and the magnitude and proportional operations can be applied to it.

Other light behaviors of the light bulb also have dependencies. The amount of the *expel* light behavior is proportional to the rate of the *create* light behavior:

```
proportional (rate [create light in light source])
```

The *destroy* light behavior gets rid of any light that does not move out of the light bulb:

```
difference (rate [create light in light source]
            rate [move light from light source to surface])
```

Expressing the depender within the value of an attribute rather than directly within a behavior is done for a good reason. Simply stating that a behavior has some dependency doesn't describe how the dependency affects the behavior. The behavior might still be active even if the dependency is not satisfied, e.g., the dependency might only affect the intensity of the behavior.

3.3.2. Tracking Dependencies to Inferred Behaviors

Suppose that instead of consolidating the battery and switch, the light bulb and switch were chosen. An *allow* electricity behavior through the light bulb-switch would be inferred, the two original *allow* electricity behaviors would be not be part of the external description of the light bulb-switch, and their connection and containers would not be referenced in the light bulb-switch's external description.

This creates a difficulty in the description of the *create* light behavior of the light bulb-switch because its dependency references places that are not part of the external description, e.g., end1 of the light bulb. Our solution is to modify the dependency so it only refers to the external description of the light-bulb switch by changing the rate quantity from:

```
proportional (magnitude (rate
     [move electricity from end1 of the light bulb
          to end2 of the light bulb
          thru electrical container of the light bulb]
```

to:

```
proportional (magnitude (rate
     [move electricity from end1 of the switch
          to end2 of the light bulb
          thru electrical container of the light bulb-switch]
```

This solution requires that the possible ways to move through the light bulb be mapped to the possible ways to move through the light bulb-switch, modifying the dependency expression as necessary (e.g., if the light bulb and switch were parallel, the fraction of electricity going through the light bulb would need to be computed).

We have not developed a complete method for tracking dependencies from components to composite components, but we have studied *move* dependencies in some detail. Since inference of a *move* behavior requires an *allow* behavior, external *allow* behaviors that move through the dependency represent the possible ways that movement through the composite component can affect the dependency. These *allow* behaviors can be identified, and the dependency can be appropriately modified by tracking the inference of *allow* behaviors from the path of the dependency to external *allow* behaviors. *Move* behaviors that go through the dependency can be similarly identified.

There is no special difficulty if several external *allows* go through the dependency. All of them can be remembered as possible ways to travel through the dependency. However, if several *moves* are inferred, there is the problem of how to add them together. Simply adding their rates together fails for two reasons:

- The rate of a *move* might be zero due to failure of implicit conditions. If the source container of a *move* is empty, or if the sink container is full, then the rate of the *move* is zero. The sum of the *moves* needs to be conditioned to take this into account.
- Since the *moves* share path segments (at least the path of the original dependency), and since their *pumps* are likely to overlap (one *pump* might be incorporated in several *moves*), the result of adding them together will probably be too high. This is because the shared *allows* and *pumps* might have an upper limit on their capacity.

Due to these difficulties, we have been unable to formulate a general way to handle situations in which several *moves* satisfy a dependency. However, there are special situations where the number of *moves* to be considered can be reduced. For example, when a set of *moves* satisfying the dependency have the same endpoints, then the *move* inferred from them based on the *parallel move* causal pattern incorporates all the influences and paths that need to be considered.

3.4. Planning Consolidation

The role of planning in consolidation is to choose what components to consolidate. In this choice, there are several goals that need to be considered:

- Minimize complexity of reasoning. Composite components with fewer external connections and behaviors are, in general, easier to reason about than composites with larger external descriptions. More connections and behaviors means that the composite components interact in more ways and along more paths. The number of behaviors cannot be predicted without using the causal patterns, bu.

the number of external connections of a composite can be determined in advance.

- Maximize information. The order in which components are consolidated can affect the amount of information that is lost. For example in figure 10, it is better to consolidate all the components in this subsystem than to consolidate any part of it with other components that are serially connected to it. In general it is better to consolidate loops than to consolidate components both in and out of the loop.
- Choose reasonable subsystems. It would be good if composite components looked like reasonable subsystems to people. With more spatial information, one could choose subsystems based on spatial closeness. This assumes that components that are closer tend to be grouped into the same subsystem. For a specific domain, there might be patterns of components that are commonly used in devices and commonly thought of as subsystems. Since our structural representation omits a lot of spatial information and since we have not concentrated on a single domain, we have not emphasized this goal.

The reduction of complexity goal is clearly the most important. In complex devices, the choice can easily lead to order of magnitude differences in the amount of work done. The main heuristic is to choose subsystems with as few external connections as possible. This heuristic also tends to satisfy the maximize information goal since consolidating loops tends to lead to subsystems with fewer external connections.

One simple way to implement this heuristic is a top-down approach, i.e., divide the device into two subsystems with a minimal number of connections between them, and recursively divide the subsystems. This is almost the same problem as minimal cut of a graph, which has a polynomial algorithm. Another simple method is bottom-up—examine the possible pairs of components that can be consolidated, and choose the pair with the fewest external connections. With minor refinements, we implemented both approaches, and both of them gave good results on the examples we modeled.

3.5. Explanation of Behavior

The primary effect of the light bulb device is that light is produced when the switch is closed. Consider now a composite that consists of the battery-switch and the light bulb. This inference can proceed as follows (figure 14 illustrates the inferences that derived the creation of light from the behaviors of the components):

- The *allow* electricity behaviors of the battery-switch (box 9 in the figure) and light bulb (box 5) satisfy the ***serial allow*** pattern, resulting in an *allow* electricity behavior around the electrical circuit (box 11). The resistance is the sum of all the individual resistances. The behavior is active only during the closed mode.
- The *pump* electricity behavior of the battery-switch (box 8) and the *allow* electricity behavior of the light bulb (box 5) satisfy the ***propagate pump*** pattern, from which a *pump* electricity behavior around the circuit is inferred (box 10). The amount of influence is equal to the amount of the battery-switch's *pump* electricity behavior. The behavior is active only during the closed mode.
- The two behaviors inferred above (boxes 10 and 11) satisfy the ***pump move*** pattern, so a *move* behavior around the circuit is inferred (box 13). The rate of the move is a function of the resistance of the *allow* behavior and the amount of influence of the *pump* behavior. The behavior is active only during the closed mode.
- The *move* electricity dependency in the *create* light behavior of the light bulb (dashed line from box 6 to box 5) is tracked from the *allow* electricity behavior of the light bulb to the *allow* electricity behavior around the circuit (dashed line from box 14 to box 11) to the *move* electricity behavior around the circuit (dashed line from box 16 to box 13).
- This *move* behavior satisfies the dependency expressed in the *create* light behavior of the light bulb (represented by dark dashed line). The rate of light creation is proportional to the the rate of electricity movement.

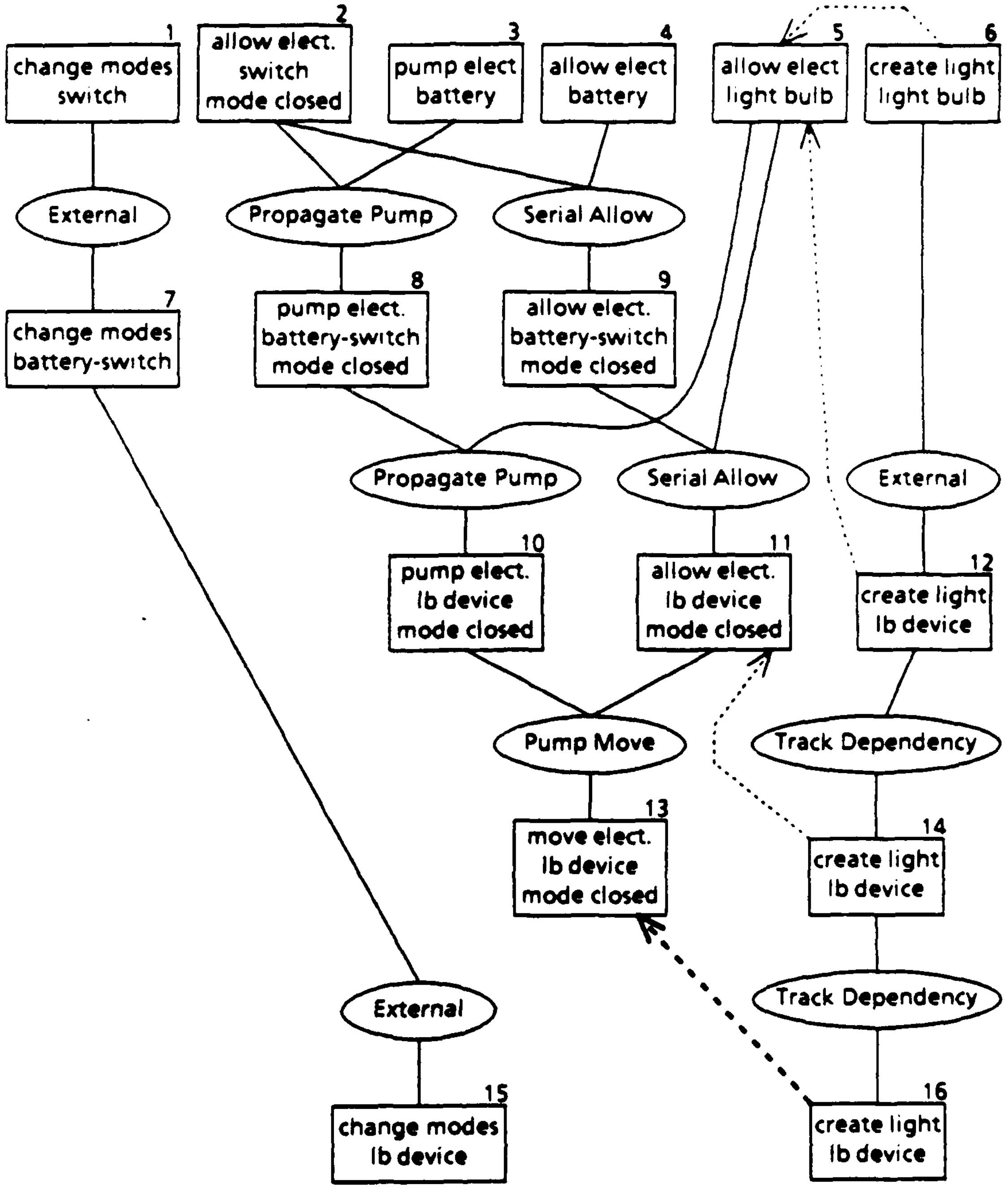


Figure 14: Inference Structure for the Creation of Light in the Light Bulb Device

This inference structure illustrated by figure 14 can be directly used as an explanation of how this device produces light. *We claim that this explanation provides a complete causal account of the creation of light in the light bulb system in terms of the components' behaviors and the device's structure.* The completeness of the explanation is not in terms of a precise analysis of the quantities, but of how the qualitative behavior of the components leads to the qualitative behavior of the device. The inference structure identifies the role of each component behavior by showing how they interact with each other to result in movement of electricity and creation of light.

Also note that because all the electrical connections are internal to the device, no electricity behavior will become part of the final description of the device's behavior. Only the signal, light, and *change mode* behaviors will be selected as external behaviors of the light bulb device. ***The device's external description states only what the outward behavior of the device is, not how it is accomplished.***

4. Example: Cardiovascular System

A model of the cardiovascular system was developed in collaboration with Jack W. Smith, Jr. and John R. Svirbely [2]. Our intent was to develop a model that could form the basis for explanation and prediction of behavior based on changes in structure and behavior. In addition, part of the purpose of this example is to illustrate some of the reasoning behind the development a behavioral model of a physical system.

Figure 15 illustrates our representation of the top level structure of the cardiovascular system. The right side of the heart moves blood into the pulmonary circulation, where the blood absorbs oxygen from and releases carbon dioxide into the lungs. The blood then flows to the left side of the heart, which pumps it into the systemic circulation, where the blood exchanges oxygen and carbon dioxide with the interstitium. Cardiovascular Control represents that part of the nervous system that regulates and synchronizes the other components Only two open connections to the lungs and the interstitium are represented.

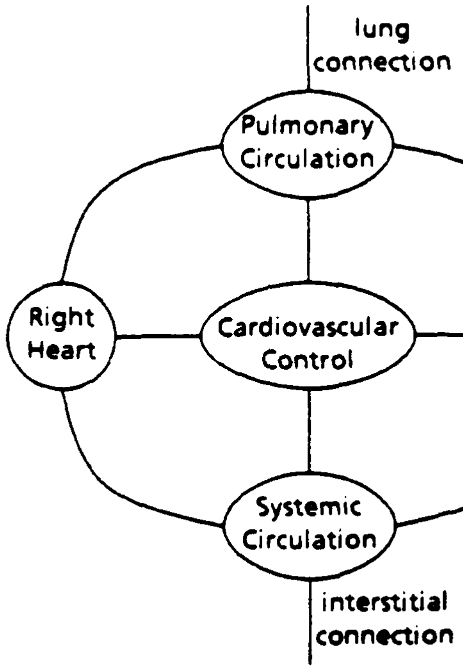


Figure 15: Structure of the Cardiovascular System

4.1. Modeling the Cardiovascular System

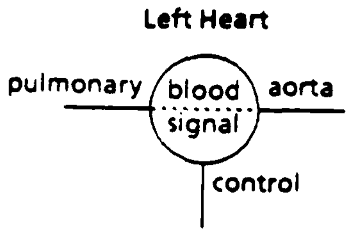
One might model the cardiovascular system with ***pump*** blood behaviors to **Right Heart** and **Left Heart**; one-way ***allow*** blood behaviors to all the components except **Cardiovascular Control**; and ***allow*** and ***expel*** signal behaviors to and from **Cardiovascular Control** so it can adjust cardiac output. However, because the **Pulmonary** and **Systemic Circulation** do not have ***expel*** blood behaviors, this model would be inadequate for any situation in which the pressure in the **Pulmonary** and **Systemic Circulation** becomes a significant factor, which is true for many cardiovascular disorders.

A more accurate description of **Pulmonary** and **Systemic Circulation** would include ***expel*** blood behaviors that depend on the amount of blood that is contained and on signals from **Cardiovascular Control** to constrict blood vessels. This would be further improved by having the behaviors of **Cardiovascular Control** depend on the amount of the *expel* blood behavior of **Systemic Circulation**, reflecting the behavior of the baroreceptors which sense the arterial pressure and pass that information to the brain.

Figure 16 is a simplified version of the **Left Heart**'s behavior in our model. The behavioral modes of the **Left Heart** are **systole** and **diastole**. The synchronization of these modes is controlled by signals coming into the **control** connection. The **Left Heart** changes from the **diastole** mode to the **systole** mode when it is signalled to do so. Changing back to the **diastole** mode occurs when **systole** is finished. Also note that the there is a *pump* blood behavior for each mode. **Amount-of-expansion-formula** and **systole-duration-formula** stand for the expressions that determines the expansion of ventricle during **diastole** and how long **systole** lasts, respectively.

The components of **Left Heart** are given in figure 17. The **Mitral** and **Aortic Valve** components have one-way *allow* blood behaviors, while the **Left Atrium** and **Left Ventricle** have two-way *allow* behaviors. The **Left Atrium** and **Left Ventricle** have *expel* blood behaviors that are regulated via the **atrium control** connection and **ventricle control** connection, respectively.

The *allow* blood behavior of **Left Heart** is caused by the *allow* blood behaviors of its components (see figure 18). Note that the one-way *allow* behaviors of the **Mitral Valve** and **Aortic Valve** makes the **Left Heart**'s *allow* behaviors to be one-way. The *pump* blood behaviors are a result of the *expel* behaviors of **Left Ventricle** (one each for **systole** and **diastole**) and the *allow* blood behaviors. The *allow* signal and *change mode* behaviors are taken from **Left Ventricle** (not shown in figure 18). Figures 16-18 are simplified as they ignore the *expel* blood behaviors and the behavioral modes of **Left Atrium**. A full account of **Left Heart**'s behavior would incorporate the *pump* behaviors caused by the **Left Atrium**, as well as the additional behavioral modes. However, it would be desirable to also derive the simpler description since it has sufficient information for many reasoning situations.



```
Connections:
    pulmonary of blood, aorta of blood
    control of signal
Containers:
    ventricle of blood, capacity positive
    nerves of signal, capacity infinitesimal
Modes:
    systole
    diastole
Behaviors:
    allow blood from pulmonary to ventricle
    allow blood from ventricle to aorta
    pump blood from ventricle to aorta, mode systole,
        amount (proportional (amount-to-contract
            [move signal from control to nerves,
                message start-systole ]))
    pump blood from pulmonary to ventricle, mode diastole
        amount amount-of-expansion-formula
    allow signal from control to nerves
    change mode to systole, mode diastole,
        when [move signal from control to nerves,
                message start-systole]
    change mode to diastole, mode systole,
        when [duration(systole) > systole-duration-formula]
```

Figure 16: Behavioral Description of Left Heart

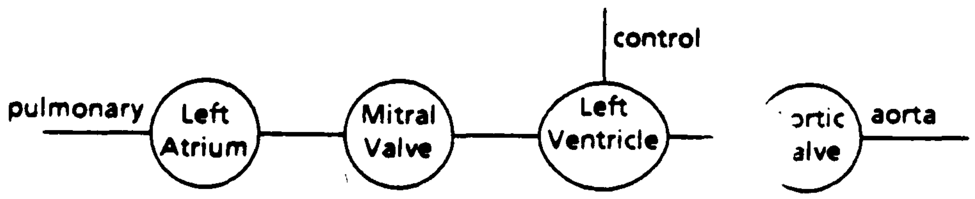


Figure 17: Components of L· .eart

Figures 19 and 20 illustrate the behavior of **Systemic Circulation** and its structure. Two connections to the interstitium for the flow of oxygen and carbon dioxide are needed because the representation and reasoning can only handle one kind of substance per connection. This is the result of an inability to represent mixtures. The second *allow* blood behavior is one-way because the **Veins** do not allow back flow. The interaction with the interstitial connection comes from the **Capillaries**. The *expel* blood behavior is a combination of the *expels* of its components, primarily the **Arteries** and **Veins**. The *expel* signal behavior arises from the **Arteries**.

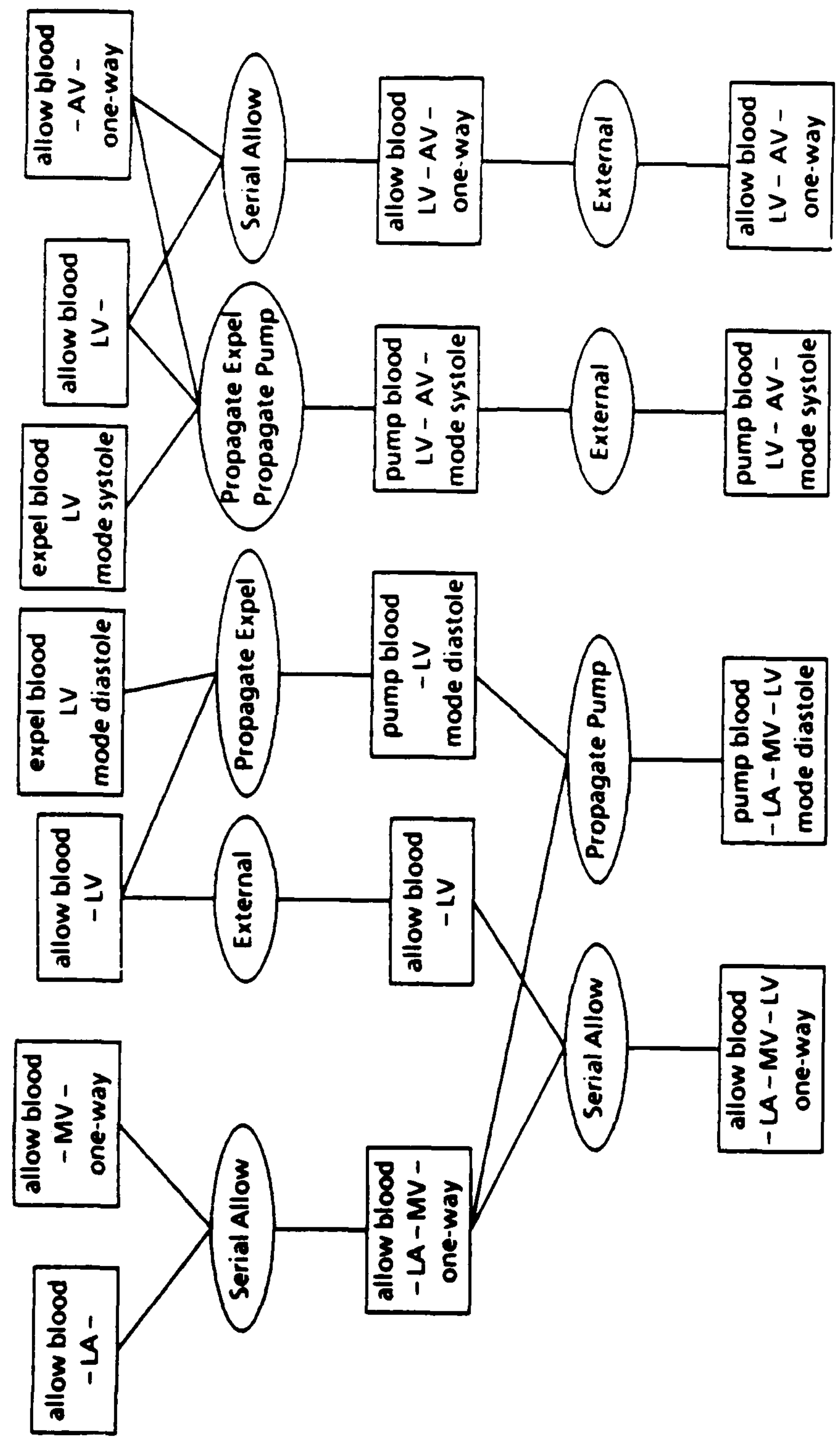
The behaviors of **Cardiovascular Control** use the signal from the **Arteries** to send signals that regulate the behaviors of the other components. For example, the signal for contracting the heart can be represented as:

```
pump signal from control-center to left heart, message contract,
    amount-to-contract (proportional (amount-of-pressure
            [move signal from systemic circulation
                  to control-center, message pressure]))
```

When the components of the cardiovascular system are considered together, the *allow*, *expel*, and *pump* blood behaviors can be used to infer *move* blood behaviors between the components and around the circulatory system. Because blood contains oxygen and carbon dioxide, the *carry move* causal pattern can be used to infer *move* oxygen and *move* carbon dioxide behaviors from the *move* blood behaviors. The paths for the flow of oxygen and carbon dioxide through the cardiovascular system are completed by connections to the interstitium (see figure 19) and to the lungs.

4.2. Modeling Hypovolemia and other Conditions

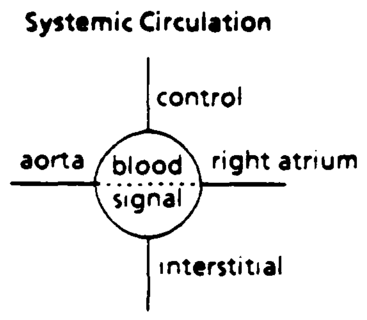
When there is a significant loss of blood, the cardiovascular system compensates in a number of ways. Some of these are directly represented, e.g., the effects of signals from **Cardiovascular Control**, while others require simulation knowledge, such as the distribution of the blood, and the size of the signals sent to the heart.


allow blood -LA-
allow blood -MV- one-way
Serial Allow
allow blood -LA-MV- one-way
allow blood -LV
External
allow blood -LV
Serial Allow
allow blood -LA-MV-LV one-way
expel blood LV mode diastole
Propagate Expel
pump blood -LV mode diastole
Propagate Pump
pump blood -LA-MV-LV mode diastole
expel blood LV mode systole
allow blood LV-
Propagate Expel Propagate Pump
pump blood LV-AV- mode systole
External
pump blood LV-AV- mode systole
allow blood -AV- one-way
Serial Allow
allow blood LV-AV- one-way
External
allow blood LV-AV- one-way

Explanation of Symbols

LA	Left Atrium
MV	Mitral Valve
LV	Left Ventricle
AV	Aortic Valve
-LV	between Mitral Valve Connection and blood container inside Left Ventricle
LV-	between blood container inside Left Ventricle and Aortic Valve Connection
-LA-	between connections of Left Atrium thru Left Atrium
one-way	from left to right

Figure 18: Inference of the *Allow* and *Pump* Blood Behaviors of Left Heart



```
Connections:
    aorta of blood, right atrium of blood, control of signal
    interstitial1 of oxygen, interstitial2 of CO2
Containers:
    vessels of blood, capacity positive
    nerves of signal, capacity infinitesimal
Behaviors:
    allow blood between aorta and vessels,
        resistance (proportional (amount-to-constrict
                          [move signal from control to nerves, message constrict]))
    allow blood from vessels to aorta,
        resistance (proportional (amount-to-constrict
                          [move signal from control to nerves, message constrict]))
    allow oxygen between vessels and interstitial1
    allow CO2 between vessels and interstitial2
    expel blood in vessels, amount
        (doubleProportional [amount vessels] (amount-to-constrict
                          [move signal from control to nerves, message constrict]))
    allow signal between nerves and control
    expel signal in nerves, message pressure, amount-of-pressure
                                    (proportional (amount [expel blood from vessels]))
```

Figure 19: Behavioral Description of Systemic Circulation

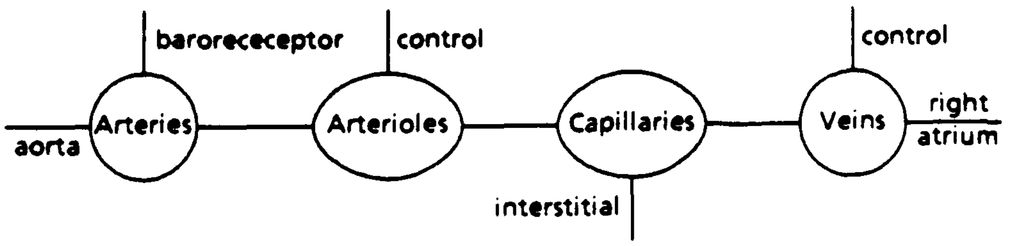


Figure 20: Components of Systemic Circulation

A hypovolemic condition (low blood volume) would result in the following propagation of effects in our model (see figure 21). First, the *expel* behaviors of the circulation components decrease since they are proportional to the amount of blood. The Systemic Circulation sends this information to Cardiovascular Control by its *expel* signal behavior and a *move* signal behavior between the two components. Cardiovascular Control then sends signals that result in (among other things) increasing the heart's contractility and increasing the resistance and pressure of the circulation. These actions maintain (if possible) the blood pressure (amount of Systemic Circulation's *expel* blood behavior).

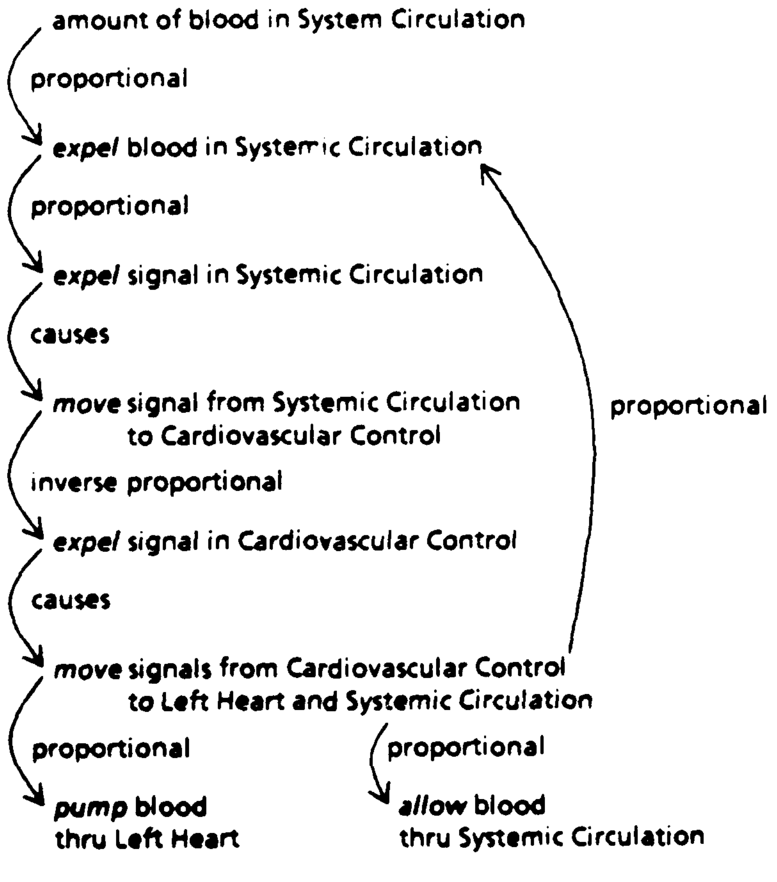

amount of blood in System Circulation
proportional
expel blood in Systemic Circulation
proportional
expel signal in Systemic Circulation
causes
move signal from Systemic Circulation to Cardiovascular Control
proportional
inverse proportional
expel signal in Cardiovascular Control
causes
move signals from Cardiovascular Control to Left Heart and Systemic Circulation
proportional
proportional
pump blood thru Left Heart
allow blood thru Systemic Circulation

Figure 21: Effects of Hypovolemia in the Cardiovascular Model

The increase in pressure is best understood by considering the components of Systemic Circulation, and what a simulation process would show. The pressure in the Arteries is proportional to the amount of blood in it. The increased activity of Left Heart moves more blood into the **Arteries** the increased resistance of the **Arterioles** and the **Veins** tends to keep more blood in the **Arteries**.

Other conditions that can be partially modeled with this representation include heart congestion and fetal circulation. If cardiac failure is represented as decreased contractility then the heart's *pump* blood behavior decreases, which results in less blood pressure Compensation by increased venous pressure raises the blood pressure in the **Pulmonary Circulation**. If we represented the fluid flow between the lungs and the **Pulmonary Circulation**, increased flow into the lungs (which can lead to pulmonary edema) would be predicted.

To represent fetal circulation (and some heart defects), additional paths for blood can be added to the representation. Consolidation can be used to determine the new behaviors of the cardiovascular components. A simulation would be required to determine how the distribution of blood and oxygen would change.

5. Weaknesses of the Current Framework

5.1. Spatial and Temporal Reasoning

In order to proceed with our research, it was necessary to oversimplify various aspects of Naive Physics reasoning, one of which is spatial representation and reasoning. The main drawback is that the structural primitives are limited to connection and containment To fully represent and reason about the behavior of complex devices, the shape and relative position of components need to be describable. It is important, e.g., to know that a piston of a car is shaped to fit inside a cylinder and to infer how the positions of the piston are constrained by the cylinder and other engine parts.

Another difficulty is the distinction between components and substances. For example, a piston acts both as a component and a substance. In its component role, it is connected to other parts of the engine and transfers energy. As a substance, the piston can be acted upon, and can move from one place to another. In general, the inability to describe movement of components, as well as the structure of substances, is a severe limitation.

The solution to these problems is to develop a spatial representation system in which shape, position, orientation, and movement can be naturally described and reasoned about, a very difficult research problem. The work of both Forbus and Stanfill is relevant this problem [13, 32]. Unfortunately, their solutions are limited in scope, as their domains are characterized by precise knowledge of shapes and very few types of substances.

We speculate that a consolidation process that builds upon Hayes's notion of "pieces of space" [18] (we use the term "space regions" below) would produce interesting results. First, there needs to be a simple representation of space regions that allows composition of regions into larger regions and maintains information about neighboring regions and relative direction, e.g. whether one region is up, down, or some other "direction" from another region. The intent of the representation would be to model space similarly to how Allen models time [1]. In analogy to Allen's work, space regions correspond to time intervals, and neighbor and direction information correspond to interval relationships. This representation could be further elaborated to describe other spatial relationships and properties, in part by borrowing ideas from vision and graphics research on representing shape (Pentland's work on representation of shape for vision is relevant here [24]). Consolidation could then proceed by combining regions, i.e. space regions correspond to components, and combinations of regions correspond to composite components. The main problems are to qualitatively describe space regions and perform consolidation based on their composition and shape.

We have also ignored temporal reasoning in our description of consolidation, mainly because temporal reasoning does not play a role in matching causal patterns. Instead, temporal reasoning comes into play in expressing the values of attributes, and in calculating the attributes of inferred behaviors. For example, if the battery of the light bulb device were replaced with AC current, the voltage would need to be described as an oscillating value. When this value is used to calculate the attributes of inferred behaviors, temporal reasoning is needed to handle this information appropriately.

5.2. Substances

Various aspects of reasoning about substances have been omitted or simplified in our framework. The most serious omission is the lack of a language for specifying knowledge about substances, i.e., for calculating attributes and handling dependencies on inferred behaviors. Qualitative arithmetics such as Simmons's [30] may lead to a solution to this problem.

One simplification is that general types of substances are not represented or reasoned about. For example, it would be more accurate to attribute a *move* fluid behavior to a pipe rather than a *move* water behavior. This would generalize the behavior the pipe, and would eliminate the need for other "kinds" of pipes for other liquid substances. This kind of problem occurred in the cardiovascular example in which different connections for oxygen and carbon dioxide were required. However, there is a difficulty with reasoning about general types of substances. When a specific type of substance is considered, how should knowledge be inherited from general types of substances and integrated with specific knowledge?

Another simplification is that mixtures of substances are not handled in a general way. Our current framework can represent some mixtures as one substance containing another, but this is inadequate for reasoning about interactions between substances. Mixtures occur in two ways. One is the mixture of different substances, such as oxygen and carbon dioxide in the blood in the cardiovascular system, and heat and light in the light bulbs in the light bulb devices. The other is the mixture of different states of the same substance, such as the gas and liquid states of the refrigerant in a refrigerator. The problem is not so much in representing that something is a mixture of various substances as in reasoning about this information. One solution is to treat mixtures as special kinds of substance with their own behavioral descriptions. The consolidation process needs to be extended to use this knowledge appropriately.

5.3. Summarization and Abstraction

The current summarization process incorporated within consolidation (composite containers and external behavior description) does not always provide a concise and efficient behavioral description of the device. Only part of the problem can be solved by more summarization processes. In addition, there is a need to abstract a single behavior out of a group of behaviors.

Failure to derive an efficient description is due to three causes. A behavioral description might have too many behavioral modes, too many structural elements, or too many behaviors (even when the first two problems have been dealt with). Having too many behavioral modes results from the fact that the devices's set of behavioral modes is logically the cross-product of the behavioral modes of the components. Two ways to reduce the number of behavioral modes are finding groups of modes that have the same behaviors. and to analyze the dependencies between behavioral modes to determine if the device prevents (due to its design) many of its behavioral modes from occurring.

If the cardiovascular system were to be described down to the capillary level of detail. there would be a problem with consolidating all the capillaries into a single subsystem—they aren't connected to each other. Consolidating groups of similar. but possibly unconnected. elements into one subsystem is one way to avoid having too many structural elements. This also requires the ability to aggregate "unconnected" behaviors. connections. and containers as well.

Even if redundant behavioral modes and structural elements are eliminated, there are still opportunities to reduce the number of behaviors of a device. For example, the function of the heart in the cardiovascular system is to pump blood between the ends of the heart. However, consolidation produces qualitatively different *pump* behaviors for each behavioral mode of each component of the heart. Also, each *pump* behavior begins from a chamber of the heart, instead of being between the connections. Recognizing that this group of *pump* behaviors can be abstracted into a single *pump* behavior requires a number of steps:

- There are *pumps* between the container and both external connections.
- The *allow* behavior between the external connections is one-way.
- The container is relatively unimportant in comparison to the *pump* behaviors, i.e., the *pumps* can move the substance through the container in a "short" period of time. The idea is that over a long enough period of time, the effect of the *pump* behaviors is more important than the storage function of the container.

This "abstraction pattern," if applicable, simplifies the behavioral description by ignoring the capacity of the container. An additional pattern could be specified to abstract a continuous *pump* from a repeating sequence of *pumps*:

- If the *pumps* occur over two or more behavioral modes, if a *pump* can be abstracted for each behavioral mode, and if there is a potential repetition of mode changes (via *change mode* behaviors), then a single *pump* can be hypothesized.

With sufficient summarization and abstraction facilities, such as the ones suggested above, consolidation would be applicable to a much larger set of devices.

5.4. Behaviors

The most serious problem with the types of behavior is the inability to represent actions at a distance, such as gravity and magnetism. Actions at a distance require causal patterns that use no structural relationship at all. For example, gravity at any point creates a *pump* mass behavior everywhere. What is needed is a new type of behavior, corresponding to action at a distance, and a globally applicable causal pattern like:

An action at a distance and any path causes a *pump* behavior

However, such an inference pattern would be untenable with any of the summarization and abstraction features currently within our framework, as well as those just discussed. This is because consolidation assumes that effects propagate structurally, i.e., that internal · ments can only be affected through external connections. One solution is to make consolidation a two-pass process in which the first pass would perform inferences that arise from actions at a distance, and the second pass would perform consolidation as usual. That is, the behavioral description of a :evice needs to be preprocessed to take care of effects that do not depend on structur· nnectedness.

A less serious problem is that *create* and *destroy* behaviors are sometimes inconvenient to use. Real Physics has conservation principles that disallow creation destruction of a substance without a corresponding destruction, creation of another. This seems to imply that a single "transform" primitive should replace *create* and *destroy*. However, Naive Physics should not be beholden to such principles. Otherwise, imaginary devices such as perpetual motion machines would not be representable.

Nevertheless, it would be convenient to be able to compose new types of behavior out of the primitive set, so that "transform" could be defined as a combination of *create* and *destroy*. This would better represent the often close relationship between these two behaviors without mandating universal conservation principles.

A more mundane improvement to *create* and *destroy* might be to have a single primitive that represents creation by a positive rate, and destruction by a negative rate. This would better model reversible processes, such as creation destruction of heat by adiabatic compression.[14]

[14]When a fixed amount of gas is compressed, it gains heat. When it is decompressed, it loses heat.

5.5. Filtering Information

In the implementation of our examples. we chose to represent certain phenomena and omit others For example, although a light bulb produces heat when it produces light. we chose not to represent this fact because reasoning about heat was not the point of the light bulb examples. While this omission was a matter of convenience as far as these examples were concerned, it is important to avoid reasoning about phenomena that are extraneous to the situation. All components, for example. contain heat. All fluids can contain dissolved substances. All spaces allow material to move through them. However. no consolidation process (or Naive Physics reasoning process for that matter) can afford to reason about all the physical phenomena and all the paths that can interact in a situation. Why, then. represent certain structural elements, and ignore others? Why reason about certain behaviors, and not others?

One answer is that consolidation must be responsive to the goals of whoever is using consolidation. Our goal, in the light bulb examples, was to show how consolidation could reason about the light bulb's *create* light behavior, so as a consequence, consolidation should reason about light. While this answer is unsatisfactory because it passes the problem to other reasoning processes, it affirms an active role for a rational agent who might consider consolidation as a means to solve a problem. The agent must recognize the elements and structure of the situation and decide which of them are important to reason about in light of the agent needs to know. Functional representations of devices might be useful for determining this kind of information [5, 15, 29].

6. Conclusion

Consolidation is possible because behavioral descriptions, as we have represented them. are *composable*; certain structural combinations of behaviors give rise to additional behaviors. These causal patterns also form the basis for additional reasoning about behavior.

The causal patterns index into knowledge about the behavior of substances. The behavioral description of composite components can be simplified by selecting those behaviors that form the external description of the composite behaviors and by grouping the containers and connections that a behavior goes through. Dependencies between behaviors might be satisfied by inferred behaviors or might need modification because of inferred behaviors.

Consolidation is only one of the reasoning methods of Naive Physics. The discovery, elaboration, and integration of these reasoning methods are important goals of Naive Physics research. Our primary contribution is the discovery of a process for performing consolidation of simple devices. We have also discussed how our current framework for consolidation needs to be extended for reasoning about more complex physical situations.

References

1. J. F. Allen. Towards a General Theory of Time and Action. *Artificial Intelligence 23*, 2 (1984), 123-154.
2. T. Bylander, J. W. Smith, Jr., and J. Svirbely. Qualitative Representation of Behavior in the Medical Domain. Proc. Fifth World Congress on Medical Informatics, IMIA, Washington, D.C., 1986. To appear.
3. B. Chandrasekaran and S. Mittal. Deep versus Compiled Knowledge Approaches to Diagnostic Problem-Solving. *International Journal of Man-Machine Studies 19* (1983), 425-436.
4. W. J. Clancey. Heuristic Classification. *Artificial Intelligence 27*, 3 (1985), 289-350.
5. R. Davis. Diagnostic Reasoning Based on Structure and Function. *Artificial Intelligence 24* (1984), 347-410.
6. J. de Kleer. Causal and Teleological Reasoning in Circuit Recognition. TR-529, Artificial Intelligence Laboratory, MIT, 1979.
7. J. de Kleer and J. S. Brown. Assumptions and Ambiguities in Mechanistic Mental Models. In D. Gentner and A. Stevens, Eds., *Mental Models*. Lawrence Erlbaum, Hillsdale, New Jersey, 1983, pp. 155-190.
8. J. de Kleer and J. S. Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence 24* (1984), 7-83.
9. J. de Kleer. How Circuits Work. *Artificial Intelligence 24* (1984), 205-280.
10. J. de Kleer and D. G. Bobrow. Qualitative Reasoning with Higher-Order Derivatives. Proc. National Conference on Artifical Intelligence, Austin, 1984, pp. 86-91.

11. P. K. Fink, J. C. Lusth, and J. W. Duran. A General Purpose Expert System Design for Diagnostic Problem Solving. Proc. IEEE Workshop on Principles of Knowledge-Based Systems, IEEE Computer Society, Denver, 1984, pp. 45-52.

12. P. K. Fink. A General Purpose Expert System Design for Diagnostic Problem Solving. Proc. Ninth International Joint Conference on Artificial Intelligence, Los Angeles, 1985, pp. 426-431

13. K. D. Forbus. Qualitative Reasoning about Space and Motion. In D. Gentner and A. Stevens, Eds., *Mental Models*, Lawrence Erlbaum, Hillsdale, New Jersey, 1983, pp. 53-73.

14. K. D. Forbus. Qualitative Process Theory. *Artificial Intelligence 24* (1984), 85-168.

15. M. R. Genesereth. The Use of Design Descriptions in Automated Diagnosis. *Artificial Intelligence 24* (1984), 411-436.

16. P. J. Hayes. The Naive Physics Manifesto. In D. Michie, Ed., *Expert Systems in the Microelectronic Age*, Edinburgh University Press, Edinburgh, 1979, pp. 242-270.

17. P. J. Hayes. The Second Naive Physics Manifesto. In J. Hobbs and R. Moore, Eds., *Formal Theories of the Commonsense World*, Ablex, Norwood, New Jersey, 1985, pp. 1-36.

18. P. J. Hayes. Naive Physics I: Ontology for Liquids. In J. Hobbs and R. Moore, Eds., *Formal Theories of the Commonsense World*, Ablex, Norwood, New Jersey, 1985, pp. 71-107.

19. B. Kuipers. Commonsense Reasoning about Causality: Deriving Behavior from Structure. *Artificial Intelligence 24* (1984), 169-203.

20. B. J. Kuipers. Qualitative Simulation. *Artificial Intelligence 29*, 3 (1986), 289-338.

21. W. J. Long. Reasoning about State from Causation and Time in a Medical Domain. Proc. National Conference on Artificial Intelligence, Washington, D. C., 1983, pp. 251-254.

22. J. C. Lusth. Diagnosis using a Functional Model. Proc. International Conference on Cybernetics and Society, IEEE Systems, Man, and Cybernetics Society, Tucson, 1985, pp. 501-503.

23. R. S. Patil, P. Szolovits, and W. B. Schwartz. Modeling Knowledge of the Patient in Acid-Base and Electrolyte Disorders. In P. Szolovits, Ed., *Artificial Intelligence in Medicine*, Westview Press, Boulder, Colorado, 1982, pp. 191-226.

24. A. P. Pentland. Perceptual Organization and the Representation of Natural Form. *Artificial Intelligence 28*, 3 (1986), 293-331.

25. H. E. Pople, Jr. Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics. In P. Szolovits, Ed., *Artificial Intelligence in Medicine*, Westview Press, Boulder, Colorado, 1982, pp. 119-190.

26. C. Rieger and M. Grinberg. The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms. Proc. Fifth International Joint Conference on Artificial Intelligence, Cambridge, 1977, pp. 250-256.

27. R. C. Schank. Identification of Conceptualizations Underlying Natural Language. In R. Schank and K. Colby, Eds., *Computer Models of Thought and Language*, W. H. Freeman, San Francisco, 1973, pp. 187-247.

28. R. C. Schank. *Conceptual Information Processing*. North Holland, Amsterdam, 1975.

29. V. Sembugamoorthy and B. Chandrasekaran. Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems. In J. Kolodner and C. Riesbeck, Eds., *Experience, Memory, and Reasoning*, Lawrence Erlbaum, Hillsdale, New Jersey, 1986, pp. 47-73.

30. R. Simmons. "Commonsense" Arithmetic Reasoning. Proc. Fifth National Conference on Artificial Intelligence, Philadeliphia, 1986, pp. 118-124.

31. H. A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Mass., 1981.

32. C. Stanfill. The Decomposition of a Large Domain: Reasoning about Machines. Proc. National Conference on Artifical Intelligence, Washington, D.C., 1983, pp. 387-390.

33. G. J. Sussman and G. L. Steele. CONSTRAINTS—A Language for Expressing Almost-Hierarchical Descriptions. *Artificial Intelligence 14* (1980), 1-39.

34. S. M. Weiss, C. A. Kulikowski, S. Amarel, and A. Safir. A Model-Based Method for Computer-Aided Medical Decision-Making. *Artificial Intelligence 11*, 1 (1978), 145-172.

35. Y. Wilks. An Artificial Intelligence Approach to Machine Translation. In R. Schank and K. Colby, Eds., *Computer Models of Thought and Language*, W. H. Freeman, San Francisco, 1973, pp. 114-151.

36. Y. Wilks. Parsing English II. In E. Charniak and Y. Wilks, Eds., *Computational Semantics*, North-Holland, Amsterdam, 1976, pp. 155-184.

37. B. C. Williams. Qualitative Analysis of MOS Circuits. *Artificial Intelligence 24* (1984), 281-346.

38. B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. Proc. Fifth National Conference on Artificial Intelligence, Philadeliphia, 1986, pp. 105-112.

A Critique of Qualitative Simulation from a Consolidation Viewpoint

TOM BYLANDER

Abstract —To understand commonsense reasoning, we need to discover what kinds of problems a commonsense reasoner should be able to solve, what the reasoner needs to have in order to solve those problems, and the relationships among the various kinds of problem solving abilities. The paper examines four methods for performing qualitative reasoning about the behavior of physical situations. Three of the methods perform qualitative simulation, which determines the behavior of a situation by a qualitative version of simulation methods. The other method is called consolidation, which derives the behavior of a situation by composing the behavior of the situation's components. The work shows that qualitative simulation and consolidation work on different problems of qualitative reasoning, and that their differences and similarities lead to several implications about their role in qualitative reasoning.

I. INTRODUCTION

A RECURRING CRITICISM of expert systems is that their knowledge is too "shallow." The criticism is directed at several problems that arise when the decisions made by these systems are based on associational knowledge (e.g., the probability of a malfunction given a datum) instead of a model of the domain (e.g., a physical description that causally relates the malfunction and the datum). Without a domain model, expert systems cannot reason about the assumptions underlying their knowledge, making errors more likely.

The reason why association-based systems continue to abound is because robust models for reasoning about complex domains do not yet exist. One area of artificial intelligence (AI) research that is directed towards the goal of more robust models is qualitative reasoning, the ability to make decisions and solve problems based on qualitative data and models. The goal of qualitative reasoning is to achieve predictive and explanatory power similar to that of quantitative and analytical models while avoiding the need for precise formulations of problems and computationally intensive methods. For example, in the situation where a flame is under a pan of water, one can predict that the water will probably heat up and boil. Even though only rough details of this situation have been described, conclusions about likely behavior can still be reached without the expense of formulating and running a quantitative model. See Gentner and Stevens [12], Bobrow [1], and Hobbs and Moore [13] for a number of articles on qualitative reasoning.

One kind of qualitative reasoning is qualitative simulation (QS) [6],[10],[15]. Like its quantitative counterpart, a description of the situation is used to determine the relevant parameters (or quantities) and constraints of the situation, a simulation is performed, and the results are transformed into interpretations of the overall behavior. Unlike quantitative simulation, specific values are not usually assigned to quantities, but only their ordinal relationship to important constants and other quantities are expressed. Also, constraints might be qualitatively stated, e.g., proportionality between two quantities might be asserted, but not necessarily a specific function. QS then tracks the situation from one qualitative state to another by predicting the changes in the ordinal relationships of the quantities based on the constraints of the situation.

We have proposed a different method called consolidation [3],[4], which is a type of qualitative analysis. The behavior of the situation is discovered by inferring the behavior of selected substructures from the behavior and interconnections of their constituents. Successive application of this process on increasingly larger substructures results in inferring the overall behavior of the situation.

QS and consolidation appear to be rival methods for the same problem. We shall show, though, that they apply to two different problems of qualitative reasoning. Thus neither QS nor consolidation fully address the problem of qualitative reasoning about physical situations. The commonalities of the two problems leads to interesting implications about the role of consolidation within a complete theory of reasoning about behavior. Most of the implications concern inherent limitations of QS and how consolidation can handle them.

Our discussion will be divided as follows. First, we briefly describe three methods of qualitative simulation. Next, we describe their commonalities and their differences. The most important commonality is their agreement on the type of problem that qualitative simulation solves. Then, after a brief description of consolidation, we characterize the differences in information processing between consolidation and qualitative simulation. Finally, these differences lead to several implications about the role

Manuscript received July 15, 1987; revised January 4, 1988. This research is supported by the Air Force Office of Scientific Research through AFOSR 87-0090, and grant NIHR GOAOE 82048-02 from the National Institute for Handicapped Research. This paper is an expanded and revised version of the one that appeared in the 1985 *Proceedings of the International Conference on Cybernetics and Society*.

T. Bylander is with the Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, Ohio State University, Columbus, OH 43210.

IEEE Log Number 8819565.

of QS and consolidation for reasoning about physical phenomena.

II. Three Methods of Qualitative Simulation

Our discussion is limited to the three most well-known qualitative simulation (QS) methods, that of de Kleer & Brown [6], [7], Forbus [10], [11], and Kuipers [15], [16], [17]. Other qualitative simulation methods, such as that of Williams [20] and Weld [19], fundamentally agree with the above methods, and so are not covered.

As part of summarizing the basic ideas and methods of each method, we illustrate how they apply to the example situation pictured in Fig. 1. In this situation, a flame is under a pan that holds some water. Both the flame and the pan are located in a room. For each method, we describe how they infer the possibility that an equilibrium occurs, i.e., that the rate of heat going from the flame to the pan becomes the same as the rate of heat going from the pan to the room.[1]

The term "quantity" is frequently used in the following discussion, so a description of quantities is in order. Quantities are used to represent the real-valued parameters of the QS. At a specific point of time in a specific situation, a quantity within that situation has a particular real value. For qualitative reasoning, though, always assigning a quantity an actual number is forbidden. Instead important real numbers and real intervals, called a quantity space (QS) [9], are identified as relevant to the quantity, and the quantity's relationship within the quantity space is its "value." In addition, the quantity's direction of change (up, constant, or down), i.e., its qualitative "derivative," is maintained for the purposes of the QS, in order to anticipate what the next value of the quantity will be. Quantities can also be used to represent qualitative derivatives so that the second, third, etc. derivatives of a quantity can be expressed.

Each of the following descriptions is necessarily too brief to completely describe each method, so much simplification has taken place. However, they should be accurate enough for the purposes of this paper.

A. The Confluences of de Kleer & Brown

This method models behavior using confluences. Roughly, confluences are qualitative equations involving quantities and their derivatives. For example, the confluence

$$[X]+[Y]=0$$

indicates a constraint on the qualitative values of the quantities X and Y. This confluence does not mean that X equals $-Y$, but asserts that the qualitative sum of their values must "include" zero. The expression "$[X]$" evaluates to the qualitative value of X, in this case, positive, negative, or zero. If a different set of qualitative values (i.e., quantity space) were desired, it would be expressed by "$[X]_Q$" where Q refers to the quantity space. Thus, the confluence states that X must have the opposite sign of Y. If X is actually 3 and Y is actually -2, this confluence is satisfied since positive plus negative is "indefinite," which includes zero. Confluences can also be applied to derivatives of quantities (∂X denotes $[dX/dt]$, the qualitative derivative of X), so one can specify how quantities move up or down in relation to other quantities. Confluences can refer to any number of quantities or derivatives, and while it is preferred that confluences use only simple addition or subtraction, other operations are allowed.

[1] We deliberately restrict the amount of detail that is represented about the physical processes which occur in this example. For example, evaporation or boiling is not represented.

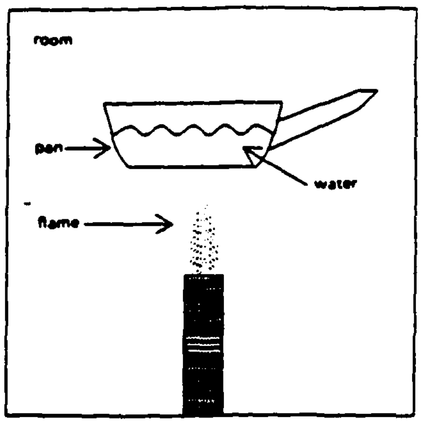


Fig. 1. Example situation of flame and pan containing water inside of room.

No intelligent agent can be expected to know in advance the set of confluences for each situation that it will experience, but it would be reasonable for an agent to know the behavior of several kinds of parts. Thus, the agent will need to describe a situation in terms of its parts, to describe the structure of a situation and the behavior of each part of the structure. For de Kleer & Brown, the elements of a situation map into disjoint components and ideal conduits between the components, called connections. Each component is modeled by a set of quantities, and a set of qualitative states. Each qualitative state is described by a condition that specifies when the component is in the qualitative state, and a set of confluences that hold during the qualitative state, i.e., the confluences describe how the component behaves in that qualitative state. Confluences and conditions on qualitative states only reference the component's quantities.

The connections indicate where material is permitted to flow from one component to another. The components of a connection specify which of their quantities are associated with the connection, and the connections are used to determine additional confluences that constrain these quantities. These confluences are used to enforce qualitative versions of general conservation laws, and provide the only means for interaction between components.

The QS is done by a method called envisioning, which is a combination of constraint satisfaction and differential perturbation. It is important to note two aspects of envisioning, one concerning the prediction of a temporal se-

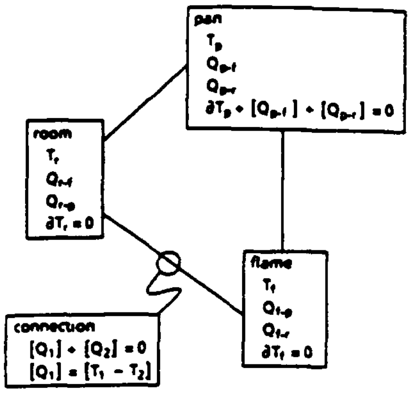


Fig. 2. Example model using De Kleer & Brown's method.

quence of events, and the other with the production of a causal explanation for the values of quantities at each moment of time. For predicting the sequence of events, simple satisfaction methods are sufficient, i.e., begin by determining values for all the quantities that satisfy the confluences, determine which quantity or quantities will next deviate from its current qualitative value, and repeat, solving the confluences (which might have changed because of a change in qualitative state) for the new values.[2]

Envisioning, however, does not simply satisfy the confluences. Instead, a quantity deviation is selected, and its effects are propagated from component to component. If there is not enough information to determine all the quantities' values, then an assumption about the value of a quantity is made based on heuristics that de Kleer & Brown have developed for explaining behavior, and the propagation continues. The path of the propagation is used to derive a causal explanation of the quantities' values.

For Fig. 1, the flame, the pan, and the room would be considered the components, and would have a connection between each pair. The water in the pan is not represented directly, but by appropriate quantities associated with the behavioral description of the pan. Heat and temperature are also not directly represented, but each component would have appropriate quantities for heat and temperature.

Fig. 2 is a simple model of this situation. Each component and connection is associated with its quantities and confluences. Each component has only one qualitative state. In a more complete description, qualitative states could be used to indicate whether the flame was on or off, or whether the water was boiling. The flame and room have ideal models of unchanging temperatures ($\partial T_x = 0$). The temperature of the pan (T_p) varies with the amount of heat flow between the pan and the flame (Q_{p-f}) and between the pan and the room (Q_{p-r}) (represented by the pan's confluence). Each connection specifies that the amount that flows from one component is the opposite of the amount that flows from the other component. The amount of the flow has the same sign as the difference in temperature.[3]

[2] This description is oversimplified because there might be ambiguity. A set of confluences might have many possible solutions and many different quantities might be the next to "deviate."

TABLE I
TEMPORAL SEQUENCE OF SELECTED VALUES IN ENVISIONING

Quantities	t_1	t_2	t_3
Q_{p-r}	0	+	+
Q_{p-f}	−	−	−
T_p	$= T_r$	$> T_r, < T_f$	$> T_r, < T_f$
∂T_p	+	+	0

Suppose that the pan and the room initially have the same temperature, which is lower than the flame's temperature. Taking the temperature of the flame as the input disturbance (imagine that it has been just turned on), then from the confluences of the connections, heat movement from the flame to the room and pan can be inferred (e.g., $[T_f - T_p]$ is positive, causing $[Q_{f-p}]$ to be positive and $[Q_{p-f}]$ to be negative). Since the room and the pan are the same temperature, there is no heat flow between them ($[T_p - T_r]$ is zero, so $[Q_{r-p}]$ and $[Q_{r-p}]$ are zero). Then from the pan's confluence, the pan's temperature must be increasing ($[Q_{p-f}]$ is negative, and $[Q_{p-r}]$ is zero, making ∂T_p positive). These values are shown in column t_1 in Table I.

In the next "episode" of time (t_2), the pan's temperature is higher than the room's because the pan's temperature is increasing, thus heat flows from the pan to the room. It is now unclear how long the pan's temperature will continue to increase, and it appears possible that the pan's temperature might start to decrease (since $[Q_{p-f}]$ is negative and $[Q_{p-r}]$ positive, any value of ∂T_p satisfies the confluence). Adding more confluences can resolve this latter difficulty. Adding $\partial Q_1 = \partial T_1 - \partial T_2$ to the connections' confluences predicts that both ∂Q_{p-f} and ∂Q_{p-r} become zero when ∂T_p becomes zero. Now adding $\partial^2 T_p + \partial Q_{p-f} + \partial Q_{p-r} = 0$ to the pan's confluences predicts that the pan's temperature stabilizes (t_3).

B. The Qualitative Processes of Forbus

Forbus introduces a notion called qualitative process (QP) to account for change and to explain why it occurs. QPs perform a similar function as confluences as they both specify behavior and interaction, but the way QPs are defined and applied is very different. First, we need to describe some of the things that QPs refer to.

Situations are composed of individuals, predicates on individuals, and relationships between them. Forbus does not provide a specific set of relationships, leaving it to the implementor to determine what relationships are relevant. An individual view is a special kind of relationship, which

[3] In this confluence, the brackets enclose the subtraction operation, which indicates that the subtraction is performed on the real values, not the qualitative values. If qualitative values were used and both temperatures were positive, then nothing could be concluded about heat flow because positive minus positive is indefinite.

```
process heat-flow

Individuals:
    src an object, Has-Quantity(src, heat)
    dst an object, Has-Quantity(dst, heat)
    path a heat-path, Heat-Connection(path, src, dst)
Preconditions:
    Heat-Aligned(path)
QuantityConditions:
    A[temperature(src)] > A[temperature(dst)]
Relations:
    Let flow-rate be a quantity
    A[flow-rate] > ZERO
Influences:
    I−(heat(src), A[flow-rate])
    I+(heat(dst), A[flow-rate])
```

Fig. 3. Heat-flow qualitative process.

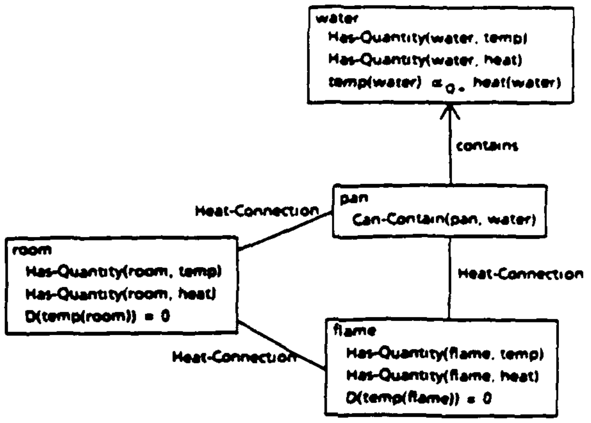


Fig. 4. Example model using Forbus's method.

consists of a set of individuals, a set of conditions that determine whether the individual view is applicable, and the relationships that follow from them. An individual view is used to "view" a group of objects as a whole. For example, a body of liquid and a container are individuals; liquid in a container satisfies the contained-liquid individual view. Similar to the components and connections of de Kleer & Brown's method, each individual and individual view is associated with a set of quantities.

The only kind of behavior description that can be directly associated with an individual or individual view is a qualitative proportionality between two of its quantities. For example, $X \propto_Q Y$ denotes a qualitative proportionality that indicates that X is dependent on Y. $\propto_{Q+}$ indicates a monotonic increasing relationship and $\propto_{Q-}$ indicates a monotonic decreasing relationship. A change in X does not necessarily imply a change in Y.

QPs are the mechanism that determines when changes occur. Unlike confluences, a QP is not part of an individual's behavioral description, but is a general rule that indicates the conditions among a group of individuals that cause an influence, an increasing or decreasing effect on the value of a quantity. For example, $I+(X, Y)$ is an influence that specifies that X is increasing at rate Y. Neither an influence nor a qualitative proportionality guarantees that a quantity actually changes in a certain direction since there might be several influences or proportionalities that affect the same quantity. The actual change in a quantity is the sum of the effects on it.

The QS works as follows: find all the individual views and QPs that are active (whose conditions are true); determine the effects specified by the influences of the QPs and indirectly by any proportionalities; determine what the change(s) will be, viz, a quantity or derivative changes to a new value, a new QP becomes active, or a previous QP becomes inactive; and repeat.

For Fig. 1, the primary QP is the heat-flow process displayed in Fig. 3. The individuals, preconditions, and quantity conditions sections in the figure specify the conditions for a heat-flow process to be active. The individuals of a heat-flow process are two objects that can store heat ("src" and "dst"), and a heat-path that connects them ("path"). The preconditions section specifies that the path be heat-aligned (meaning that there is nothing blocking the flow of heat along that path). The quantity conditions section requires that the temperature of "src" ("A" is a function that refers to the amount of a quantity) be greater

TABLE II
TEMPORAL SEQUENCE OF SELECTED VALUES IN QUALITATIVE PROCESS

		t_1	t_2	t_3
Heat-flow Processes	flame to water	active	active	active
	flame to room	active	active	active
	water to room	inactive	active	active
	flow rate			
	flame to water	+	+	+
Quantities	flow rate			
	water to room	0	+	+
	D(heat(water))	+	+	0
	T_w	$= T_r$	$> T_r, < T_f$	$> T_r, < T_f$

than the temperature of "dst." The relations section specifies additional relations that hold while the process is active. In this process, a quantity called "flow-rate" is created which is greater than zero. The influences section specifies the effects on quantities. In this case, there is a negative effect on the amount of src's heat, and a positive effect on dst's heat. The amount of this effect is the amount of flow-rate.

The situation in Fig. 1 can be modeled with the flame, the room, the pan, and the water as objects with heat-paths between the flame, room, and pan (see Fig. 4). The flame, room, and water each has quantities of heat and temperature. The temperature of the water is proportional to the amount of its heat. Again, assume that the temperature of the room and the flame remains constant, the flame is hotter than the room, and the room and water are initially the same temperature. We assume that heat-paths to the water are inferred from the contained-liquid individual view.

Initially, two heat-flow processes are active, from the flame to the room and from the flame to the water (refer to Table II). The amount of the water's heat increases (D(heat(water)) is positive), which because of the proportionality, implies that the water's temperature increases. In the next time "interval," (t_2), the temperatures of the water and room are different, so a heat-flow process from the water to the room becomes active. Now the same problems as before reappear. It is questionable how long the water's temperature will continue to increase (one heat-flow process has an increasing effect, and the other

quantities:	T_f	(temperature of the flame)
	T_p	(temperature of the pan)
	T_r	(temperature of the room)
	ΔT_{p-r}	(temp. difference between pan and room)
	ΔT_{f-p}	(temp. difference between flame and pan)
	Q_{p-r}	(heat flow from pan to room)
	Q_{f-p}	(heat flow from flame to pan)
	Q_p	(net heat flow to pan)
constraints:	$IQ(T_r) = std$	(T_r is steady/constant)
	$IQ(T_f) = std$	(T_f is steady/constant)
	$T_p = T_r + \Delta T_{p-r}$	(defines ΔT_{p-r})
	$T_f = T_p + \Delta T_{f-p}$	(defines ΔT_{f-p})
	$Q_{p-r} = M_z^+(\Delta T_{p-r})$	(heat flow is proportional
	$Q_{f-p} = M_z^+(\Delta T_{f-p})$	to temperature difference)
	$Q_{f-p} = Q_{p-r} + Q_p$	(defines Q_p)
	$Q_p = dT_p/dt$	(relates net heat flow to temperature)

Fig. 5. Example model using Kuipers's constraints.

has a decreasing effect), or whether it even decreases at a later point in time. To avoid the latter problem, the heat-flow process needs to be modified so that the flow-rate is proportional to the temperature difference, and that the flow rate approaches zero as the temperature difference approaches zero. With this modification, if and when the derivative of the water's temperature becomes zero, then the derivatives of all the flow-rates become zero, and the situation stabilizes.[4]

C. *The Envisionment Method of Kuipers*

Kuipers begins at a different point than either de Kleer & Brown or Forbus. Instead of modeling the individual objects of a physical situation and showing how the device model can be obtained from the structural relationships between the objects, Kuipers directly represents the quantities of the situation and the constraints between them.

The "structural description" of Kuipers's method consists of a set of constraints on a set of quantities. The constraints specify numerical relationships between the quantities. $X = Y + Z$ indicates that X equals exactly Y plus Z, and consequently, also indicates how their derivatives are related, e.g., if Y and Z are increasing, then so is X. $Y = M^+(X)$ indicates that Y is a monotonic function of X. $Y = dX/dt$ indicates that Y is a qualitative derivative of X. Kuipers's constraints are similar in spirit to de Kleer & Brown's confluences, in that both of them specify operations on quantities and a method for testing qualitative equality. However, addition, multiplication, and other arithmetic operators have their normal arithmetic meaning.

Fig. 5 is a model of Fig. 1 using this method. One bit of notation needs to be explained—M_z^+ indicates that when one quantity is zero, both quantities are zero.

Suppose again that the temperature of the flame is higher than the temperature of the room, and that the pan's temperature starts out the same as the room's. From the constraints, there is heat flow from the flame and the pan, but none between the pan and the room, thus the temperature of the pan is increasing (last constraint). Because the "+" constraint allows inferences about derivatives in Kuipers's system, it can be inferred that ΔT_{p-r} increases (constraint 3), and ΔT_{f-p} decreases (constraint 4). In addition, Q_{p-r} is increasing (constraint 5), Q_{f-p} is decreasing (constraint 6), and Q_p is decreasing (see Table III).

TABLE III
TEMPORAL SEQUENCE OF SELECTED QUANTITIES IN KUIPERS'S ENVISIONMENT

Quantities	t_1	t_2	t_3
Q_{p-r}	0	+	+
Q_{f-p}	+	+	+
T_p	$= T_r$	$> T_r, < T_f$	$> T_r, < T_f$
$IQ(T_p)$	pos	pos	std

At the next time "point," the temperature of the pan rises towards the flame's temperature. Q_{p-r} is positive and increasing and Q_{f-p} is positive and decreasing, so Q_p will continue to decrease towards zero. When Q_p becomes zero, then T_p will be steady, which leads to steady values for the temperature differences and the heat flows.

III. COMMONALITIES AND DIFFERENCES IN INFORMATION PROCESSING BETWEEN THE QUALITATIVE SIMULATION METHODS

The three QS methods summarized above represent physical situations in remarkably similar ways. In particular, they agree that a certain kind of constraint is appropriate for representing behavior and that some form of constraint satisfaction is needed for performing the simulation. The three proposals differ on the relationship between physical situations and constraints, but there are key points of agreement on the kinds of information that qualitative simulation uses and produces. Many of these similarities and differences have been pointed out by de Kleer & Brown [5] and Bonissone & Valavanis [2].

A. *Commonalities of Architecture*

The different QS methods fundamentally agree on the constraint architecture that underlies qualitative simulation, i.e., they agree on a general computational mechanism in which constraints and conditions on constraints are used to specify what "computations" can take place. Specifically, they agree on the following features.

One common feature is to model time in qualitative units, which we term "time segments." A time segment is used to model an instant or interval during which the physical situation is in a particular state. The passage of time is modeled by a sequence of time segments.

Another common modeling construct is the quantity, which was briefly discussed earlier. For each time segment, a quantity has a qualitative value, which is a real number or a real interval. The change in direction of the quantity's value is specified by a qualitative derivative, which can be another quantity. The possible values of a quantity are specified by its quantity space [9]. Quantities are used to correspond to the parameters of a physical situation, and the quantity spaces are used to specify the important values and the intervals between them.

[4]It is not clear whether Forbus's system can currently perform this analysis, but it is easy to imagine that it can be modified to do so.

TABLE IV
SUMMARY OF COMMONALITIES OF ARCHITECTURE IN QUALITATIVE SIMULATION METHODS

Commonality	de Kleer & Brown	Forbus	Kuipers
time segment	episode	interval	time-point
quantity	quantity	quantity	parameter
QDC	confluence	relation, influence and proportionality	arithmetic, functional and derivative constraints
condition	qualitative state	individual views and processes	inequality and conditional constraints

A third common feature is the use of qualitative differential constraints (QDC). A QDC describes an arithmetic relationship among a group of quantities and derivatives. The notation of a QDC can be that of an equation, such as the confluences of de Kleer & Brown and the constraints of Kuipers, or of a rule, such as the qualitative proportionalities and influences of Forbus. QDCs correspond to the physical interactions within and among elements of a physical situation.

Different states of a situation might have different quantities and QDCs that apply to it. Conditions on the applicability of quantities and QDCs are another common feature of qualitative simulation methods. All the methods provide for arithmetic tests on quantities. Forbus allows for additional conditions based on properties and relationships within a physical situation.

A fourth common feature is the process of qualitative simulation. It proceeds by performing the following steps (not necessarily in sequence) for each time segment.

- The physical situation is mapped into a set of quantities and a set of QDCs over those quantities. The initial situation specifies the values of some of the quantities.
- Constraint satisfaction is performed to validate those quantities that are bound, and determine the possible values of quantities that are not. A failed constraint indicates that this physical state cannot occur. Any physical state that always leads to an impossible-to-occur state is also considered impossible to occur.
- Differential perturbation (modifying quantities in accordance with their derivatives) is performed to discover what changes in the qualitative values of quantities might happen next. These changes are used to generate the possible states of the situation in the next time segment.

The simulation ends when no more changes occur or no new states are produced (e.g., an oscillation repeats the same physical states).

Table IV summarizes the commonalities listed above and the terminology used by each method.

B. Differences in Representing Physical Situations

How can a physical situation be mapped into the QDC architecture? Each of the QS methods has a different answer to this question. De Kleer & Brown and Forbus first represent the physical structure of the situation, and then determine the quantities and QDCs based on the elements of the situation and the structural relationships between them. Kuipers does not represent physical structure, but instead, directly lists the quantities and constraints.

De Kleer & Brown represent a physical situation by specifying its components, and the connections between the components. The behavior of each type of component and connection is represented by a set of quantities and QDCs. Their notion of qualitative state provides a way to attach conditions on QDCs. Interaction between components and connections is modeled by equating quantities of connections with quantities of the components that they connect. A few additional confluences are created to constrain groups of connections in order to enforce qualitative versions of conservation laws.

Forbus represents a physical situation by specifying its individuals and their properties and relationships. Forbus does not provide a primitive set of relationships, but on the basis of his examples, connection and containment are included as relationships. Each individual specifies its quantities and proportionalities. Individual views and qualitative processes specify how groups of individuals interact under certain conditions. Individual views can specify additional quantities and proportionalities that apply to the individuals. Qualitative processes specify influences and other relations that result from the process, such as additional quantities and changes to the structure of the situation.

Kuipers does not pr[illegible] to map from physical structure to QDCs. Inst[illegible]ad, [illegible] structural description" directly specifies the quantities and the constraints of the situation as a whole.

C. Commonality of Information Processing

Despite these differences in representing physical situations, some abstract points of agreement can be pointed out. All three proposals agree on the nature of the output of QS—a temporal sequence of physical states that the situation goes through with an explanation of why it happens.[5] There are differences over what the output includes (e.g., Forbus includes structural changes while the others do not), and over what explanation is (e.g., de Kleer & Brown base their explanation on a special

[5]In general, the ambiguity introduced by the qualitative representation and reasoning might lead to ambiguity about what sequence of states occur, so be to accurate, the output is all the possible sequences of states. Also, each sequence generated should have an explanation.

constraint propagation technique). However, they all generate physical states of the situation, assert temporal relationships between the states, and give reasons for making those conclusions.

In addition, de Kleer & Brown and Forbus agree on the nature of the input of QS. Both methods represent the structure of a physical situation by describing its parts and the relationships between the parts, and associate behavioral knowledge with the parts and their structural relationships. Although they differ on how to represent structure and how structure maps to behavior, they at least agree on what needs to be done. This agreement doesn't apply to Kuipers since his method ignores physical structure. However, Kuipers doesn't propose any alternative to how quantities and QDCs come from physical situations.

IV. Our Consolidation Method

The consolidation method uses a structural model similar to de Kleer & Brown's, adding an additional structural relationship called containment. Things like water and heat are then modeled as substances that are contained by the components of the situation, or perhaps other substances, e.g., water contains heat.

Behavior is modeled by specifying the actions (loosely called behaviors) that are performed on substances. Possible types of behaviors include the following.

- *Allow*. Permit movement of a substance from one place to another.
- *Pump*. Attempt to move a substance through a path.
- *Expel*. Attempt to move a substance from (or to) a location.
- *Move*. Move a substance from one place to another.
- *Create*. Create a substance at a location.
- *Destroy*. Destroy a substance at a location.

Each behavior specifies the kind of substance that is affected, and the location(s) (containers and connections) where it takes place. Each behavior and location may have a number of substance-specific parameters or quantities. The value of a parameter may be real, but is not restricted to be so, e.g., the "rate" of a move signal behavior might be "on" or "off." A quantity may refer to parameters of other behaviors, indicating that the behavior is dependent on other behaviors, e.g., the amount of the create light behavior of a light bulb is dependent on the rate of a to-be-inferred move electricity behavior through the light bulb.

A component is modeled by specifying its structure, i.e., its containers and potential connections to other components, and the behaviors which take place within that structure. A component may have several behavioral modes, each of which is associated with a different set of behaviors. Change mode behaviors indicate the conditions under which one mode changes to another.

Consolidation gets its name from the processing that this method proposes. The major processing sequence is to instantiate a composite component consisting of two components (either or both can themselves be composite components), and then to infer the behavior of the composite from the behavior of its subcomponents. Performing consolidation on increasingly larger composite components results in inferring the behavior of the whole situation. As a byproduct, a hierarchical behavior structure is produced that explains how the overall behavior is caused by the components' behavior.

serial allow:	*allow* substance S along path P_1
	allow substance S along path P_2
	P_1 is serial with P_2
	$\Rightarrow$ *allow* substance S along serial combination of P_1 and P_2
propagate expel:	*expel* substance S from location L
	allow substance S along path P
	L is the beginning of path P
	$\Rightarrow$ *pump* substance S along path P
include expel:	*expel* substance S from location L
	pump substance S along path P
	L is an endpoint of path P
	$\Rightarrow$ *pump* substance S along path P (incorporating effects of *expel* behavior)
propagate pump:	*pump* substance S along path P_1
	allow substance S along path P_2
	P_1 is serial with P_2
	$\Rightarrow$ *pump* substance S along serial combination of P_1 and P_2
pump move:	*pump* substance S along path P
	allow substance S along path P
	beginning of path P is a source of substance S
	end of path P is a sink of substance S
	path P has no internal sinks or sources of substance S
	$\Rightarrow$ *move* substance S along path P
parallel move:	*move* substance S along path P_1
	move substance S along path P_2
	P_1 is parallel with P_2
	$\Rightarrow$ *move* substance S along parallel combination of P_1 and P_2

Fig. 6. Example causal patterns in consolidation.

Behaviors of composite components are inferred based on composition rules, called causal patterns, each of which describes how one type of behavior can arise from a structural relationship between certain types of behavior. The existence of an inferred behavior is confirmed, and its parameters are determined using knowledge about the behavior of the substance being acted upon. Roughly then, causal patterns are used to infer aggregate behaviors, and once a pattern is matched, substance-specific knowledge is called upon to fill in the behavior's parameters. Fig. 6 illustrates several causal patterns. For example, the *serial allow* causal pattern states that if two *allow* behaviors are in series, then infer an allow behavior over the whole path. Knowledge about the substance involved might be used to determine parameters such as resistance.

The components of Fig. 1 would be the flame, pan, and room, which would all be connected to one another. The room and flame can be modeled as containers of heat (or containers of some other substance that contains heat) with *expel* heat behaviors (see Fig. 7). The pan contains water, which in turn contains heat and has an *expel* heat behavior. The temperatures of the flame, water, and room are represented by the amounts of the *expel* heat behaviors. Each component has *allow* heat behaviors between its connections and containers.

Consolidation infers that there are *move* heat behaviors between all of the components with the amount of movement proportional to the difference between temperatures (amounts of the *expel* heat behaviors). The reasoning that infers heat movement between the flame and the pan is as

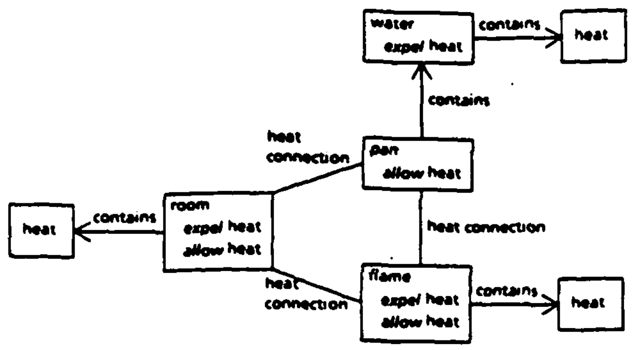


Fig. 7. Example model using consolidation.

follows.

1) The *serial allow* causal pattern is used to infer an *allow* heat behavior between the flame and the pan:
   allow heat between the "heat container" of the flame and the flame-pan connection (behavior of flame)
   allow heat between the flame-pan connection and the "heat container" of the water (behavior of pan)
   ⇒ *allow* heat between the heat containers of the flam' and water.
2) The *propagate expel* causal pattern infers a *pump* heat behavior between the flame and the pan:
   expel heat from the heat container of the flame (behavior of flame)
   allow heat between the heat containers of the flame and water (inference 1)
   ⇒ *pump* heat from the heat container of the flame to the heat container of the water.

   The amount of the *pump* behavior is proportional to the amount of the *expel* behavior.
3) The *include expel* causal pattern combines the just inferred *pump* heat behavior with the *expel* heat behavior of the water.
   expel heat from the heat container of the water (behavior of water)
   pump heat from the heat container of the flame to the heat container of the water (inference 2)
   ⇒ *pump* heat from the heat container of the flame to the heat container of the water, incorporating effects of the *expel* behavior.

   The amount of this *pump* behavior is proportional to the difference between the amounts of the *expel* behaviors of the flame and water.
4) The *pump move* causal pattern is used to infer a *move* heat behavior between the flame and the water.
   pump heat from the heat container of the flame to the heat container of the water (inference 3)
   allow heat between the heat containers of the flame and pan (inference 1)
   ⇒ *move* heat between the heat containers of the flame and pan

   The rate of the *move* behavior is proportional to the amount of the *pump* behavior, which is proportional to the difference of the amounts of the original *expel* behaviors.

The other *move* heat behaviors (between the flame and the room, between the water in the pan and the room) are inferred in a similar manner.

V. Differences in Information Processing between Consolidation and Qualitative Simulation

Consolidation, unlike any of the QS methods, does not produce a temporal sequence of physical states as its output, yet each method claims to derive the behavior of a situation. Each method starts from similar models of the situation, and makes inferences about behavior, so how can the final result, conclusions above the situation's behavior, be different? Our answer is that consolidation provides a different sort of conclusion about behavior then QS does. QS and consolidation solve two different problems.

A. Two Kinds of Behavior

Part of the confusion comes from the fact that "behavior" is an ambiguous word, and that QS and consolidation pinpoint two of its meanings. This distinction can be seen in the differences between the behavior that is input to QS and the behavior that QS outputs. The behavior that QS outputs is a temporal sequence of states that are predicted to occur in the physical situation.[6] We use the term "actual behavior" to describe this sense of behavior.

The behavior that is input to QS is somewhat harder to characterize. A QDC does not predict a temporal sequence of states, but it does restrict what states can occur. Although a single QDC is not sufficient to assign qualitative values to quantities, it does assert how the quantities affect one another. This kind of behavior is not a prediction of what will happen, but can be used for prediction if additional information is given. We call this sense of behavior "potential behavior."

The input and output of consolidation also do not describe a temporal sequence of states, but are concerned with the actions (types of behavior) that occur in a physical situation. The representation is primarily about the structures where these actions take place and the substances that are being acted upon. For example, *allow* behaviors describe which paths permit movement of which substances. The parameters of actions provide an additional level of detail. For example, the *move* heat behavior between the flame and the water in the pan does not specifically assert when heat moves, but that the situation is ripe for heat movement to occur, and that the rate of heat movement can be calculated if some other facts are known, in this case, the temperatures of the flame and water. Even if these temperatures were known at a given

[6]More than one temporal sequence of states might be output because of ambiguity; nevertheless, QS attempts to predict what will happen.

instant, more information is needed to determine whether the objects' temperatures will increase or decrease. Other external sources of heat (e.g., the room) will affect their future temperatures, so future temperatures and heat flow cannot be predicted without taking into account the entire situation. The *move* heat behavior is an indication of what potentially can happen (heat flow), and points to other information on which this potential is dependent (temperatures). Thus, the types of behaviors of consolidation are also a description of potential behavior, rather than actual behavior.

B. The Information Processing Tasks

The information processing task of a problem is a functional specification of the problem in information terms, i.e., the information that the input and output represent. This specification is part of what Marr calls the computational theory of an information processing task [18]. Above, we identified what the behavioral inputs and outputs of QS and consolidation are, so the tasks can now be specified.

The information processing task of de Kleer & Brown's and Forbus's version of QS is:

input:	physical structure of situation, and potential behavior of elements
output:	actual behavior of situation

For Kuipers, the task is:

input:	potential behavior of situation
output:	actual behavior of situation

The common feature is that all versions of QS go from potential behavior to actual behavior. De Kleer & Brown and Forbus make more commitments than Kuipers concerning the composition of physical situations, and how behavioral knowledge is associated with their structure.

For consolidation, the information processing task is:

input:	physical structure of situation, and potential behavior of elements
output:	potential behavior of situation

The key difference between QS and consolidation is that consolidation shows how descriptions of potential behavior can be derived. Because actual behavior is more specific than potential behavior, one would expect that qualitative simulation would require more specific input or additional processing time than consolidation would. In Sections VI-A and VI-B, we point out what additional information qualitative simulation needs and in Section VI-C we argue that qualitative simulation requires a global reasoning process for each time step.

C. Understanding Physical Behavior

What does it mean to understand the behavior of a situation? The two tasks have different views, and would appear to argue against each other as follows. The QS side would say that understanding behavior means being able to predict events and determine temporal relations between events. A model of behavior is useless unless it can be used to predict what happens.

The consolidation side would grant that determining what happens is important, but would note that QS works because the elements of a situation are well understood, i.e., QS is given a model of their potential behavior. However, QS doesn't provide a similar understanding of the situation because it doesn't produce a model of the situation's potential behavior.

Both sides of this "argument" miss a plausible compromise. Neither task incorporates a complete understanding of physical behavior, e.g., neither task takes on the problem of designing physical devices. Understanding, then, does not consist of being able to solve a single information processing task, but in applying a range of problem solving abilities to complex problems. QS and consolidation can be viewed as different modalities of understanding behavior. For some problems, QS might be the primary modality, while in others, consolidation might be, while still yet others, both QS and consolidation might be secondary, perhaps not even needed at all. Sections VI-D and VI-E look at two aspects of understanding: knowledge representation and causal explanation.

VI. Implications of the Differences in Information Processing

The differences in the information processing tasks between QS and consolidation can be used to further understand the QS and consolidation methods.

A. Consolidation Does Not Need Initial Conditions

Suppose that in Fig. 1, no initial conditions (initial values of quantities) were known, but some statement about the situation's behavior is still desired. Without initial conditions, QS is unable to start. The best that could be done would be to enumerate all the possible initial conditions and perform QS on each possibility. An enumeration of initial states would be small in the simple case of Fig. 1, but in more complex situations, there would be many possible initial states. Thus a better characterization of QS's information processing task would be

input:	physical structure of situation, potential behavior of elements, and initial conditions of situation.
output:	actual behavior of situation.

Consolidation can proceed without assuming any initial conditions, and in fact, the processing described earlier did not do so. If we examine more closely what some of the final results looked like (Fig. 8), it is not hard to see why this is the case. Each quantity is defined not in terms of specific values at specific moments of time, but in terms of how it is dependent on other quantities. Thus if the water in the pan happened to be hotter than the flame, then the

```
rate[move heat from flame to pan]
     = proportional( amount[expel heat from flame]
                     - amount[expel heat from pan])

rate[move heat from flame to room]
     = proportional( amount[expel heat from flame]
                     - amount[expel heat from room])

rate[move heat from pan to room]
     = proportional( amount[expel heat from pan]
                     - amount[expel heat from room])

amount[expel heat from pan]
     = proportional( amount[heat within pan])
```

Fig. 8. Some results of consolidation in flame, pan, and room example.

rate of heat flow from the flame to the water would be negative, indicating that heat would flow from the water to the flame. Fig. 8 is a condensed representation of potential behavior that can be directly used to answer questions and for other purposes, including QS if the initial state is known.

B. Consolidation Handles Open Systems

A similar problem for QS is deriving the behavior of situations that are open systems, i.e., there is interaction between the situation and the outside world. Without knowledge of what these interactions are, the value of each quantity that can be affected becomes indeterminable. Enumeration of all conceivable outside interactions is not, in general, a feasible solution since the number, kind, and order of interactions can vary greatly. However, the ability to reason about open systems seems to be necessary for understanding behavior since most situations that an agent could be expected to encounter are open systems, and parts of situations are by definition open systems. Because of this, our description of QS's information processing task can be further refined as follows.

input:	physical structure of situation,
	potential behavior of elements,
	initial conditions of situation, and
	outside interactions.
output:	actual behavior of situation.

By providing a concise representation of potential behavior, consolidation gives a solution to describing the behavior of open systems. If we changed the model of the room in our example so its temperature could fluctuate, and it had a potential "heat connection" to the outside, the result of consolidation would not be fundamentally changed. The only difference is that the room could gain or lose heat through other interactions. Heat moves among the room, flame, and pan in pretty much the same way.

The refinement of QSs information processing task also leads to a clearer distinction between potential behavior and actual behavior. Ideally, potential behavior incorporates no assumptions about initial conditions or outside interactions, while actual behavior must incorporate such assumptions for reasons stated above. In practice, however, several assumptions are made when the potential behavior of a real situation (Fig. 1) is represented (Figs. 2, 4, 5, and 7), e.g., attributing two connections to the flame. Thus, it might be best to think of the actual vs. potential distinction as a continuum, rather than a binary choice.

C. Qualitative Simulation Is a Global Reasoning Process

To perform the simulation for a particular moment in time and to check if it has been done consistently, all the elements of the situation must be taken into account. For example, the derivative of every quantity must be examined to update the quantities' values. This is true no matter the number of quantities and QDCs the situation model has. The nature of QS prevents a hierarchical breakdown since any part of a situation is very likely to be an open system, to which QS cannot be applied.

Integrating consolidation with QS might help alleviate this difficulty. Consolidation could be used to determine the potential behavior of the situation or subsituations, and QS can then be applied to the result of consolidation. In other words, even if a temporal sequence of states is the desired output, consolidation can be used to reduce the apparent complexity of QS.

D. Consolidation Places Additional Constraints on Representation

Both consolidation and QS have the same kind of input, so representations of potential behavior should be amenable to both kinds of problem solving. From the consolidation point of view, representations should facilitate the composibility of behaviors. The representations of the QS methods do not have this property.

In de Kleer and Brown's representation, consolidation would need to derive the confluences and quantities of composite components from the confluences and quantities of individual components. The task is not as simple as concatenating all the behavioral descriptions of all the parts. Instead, there is a need to directly specify how the subsystem interacts with the outside world. Consequently, an analysis of constraints would be required, so that all the constraints that apply to the subcomponents are reduced to a more perspicuous set of QDCs that expresses the composite's behavior. This is an open and difficult issue. Dormoy has discovered a substitution technique [8] that can simplify certain kinds of confluences, for example:

$$[W]=[X]+[Z]\wedge[W]=[Y]-[Z] \Rightarrow [W]=[X]+[Y].$$

However, substitution is not promising as a general technique. For example, suppose that $[W]=[Y]+[Z]$ and $[X]=[Y]+[Z]$. However, $[W]=[X]$ does not follow because if $[Y]$ is negative and $[Z]$ is positive, then W and X can have different signs without any contradiction. This difficulty in simplifying confluences would make it hard to use consolidation on this representation.

The initial difficulty for consolidation with respect to Forbus's representation is that composing behaviors doesn't make any sense. Processes, not individuals, specify the direct effects that take place. The alternative is to use individual views to specify composite components so that processes correctly apply to them, i.e., by giving composite components the right quantities and relationships so that they satisfy the conditions of appropriate processes. Doing this requires something isomorphic to the causal patterns and the substance knowledge that consolidation currently uses. For example, to derive the voltage quantity for two batteries in series, we need to know that the batteries and the composite each has its own voltage quantity (a *pump* electricity behavior is caused by two *pump* electricity behaviors in serial), and that in this kind of configuration, voltage is additive (the electrical knowledge that is invoked when the *serial pump* pattern is satisfied). So Forbus's representation has no special advantages, and would actually obscure the underlying regularity (the *serial pump* causal pattern). Another difficulty is when a process occurs inside the composite component, e.g., heat moves within the flame-pan composite.

E. Consolidation and Qualitative Simulation Provide Different Causal Explanations

Two kinds of causal explanation correspond to the two senses of behavior defined earlier. QS emphasizes the causality of temporality and propagation, i.e., the current state of the situation leads to the next, and the value of one quantity changes (via some QDC) the value of another quantity. Consolidation emphasizes the causality of composition, i.e., the behavior of a group of components arises from the behavior and structure of the individual components (see Iwasaki and Simon [14] for a third sense of causality). Another debate like the device understanding debate could be promulgated at this point with probably the same result. Neither kind of causality is necessarily superior to the other, but their usefulness depends on the particular problem to be solved. It is worthwhile to note that there can be causal explanations of situations with unknown initial conditions and in open systems (consider two batteries connected in series). Consolidation can be used to point out this aspect of causality.

F. Caveats

Currently, consolidation is not able to model all the phenomena that the QS methods are able to. Also, there may be certain kinds of situations in which consolidation would not work very well.

Forbus's method is able to handle spatial reasoning that is more complex than the connection and containment variety. For example, he can model, to a limited extent, motion and collision of objects. Forbus can also model properties of material, changes in material, and changes in structure. Consolidation has not been extended to cover these phenomena.

One possible problem for consolidation is constructing a plan for composing components in a sensible way. We are developing general heuristics for making this decision, e.g., select composite components with as few outside connections as possible. Also, domain knowledge could provide additional heuristics. In many cases, this will not be an issue because the structural hierarchy will be known in advance.

The analysis that consolidation provides holds only as long as the components remain connected in the same way. If a connection (or a component, for that matter) is created or destroyed, then much of the analysis will be invalid and must be redone. This need for reanalysis would make consolidation poor on situations where the structure frequently changes.

Currently, consolidation is dependent on dividing the elements of a situation into components and substances. In some situations, this division cannot be strictly applied. It is usually reasonable to model a light bulb, for example, as a component, but to be able to reason about a ball colliding with the light bulb, the light bulb would need to be modeled as a substance that is contained by the space it is in, and the ball is trying to get into the same "container." Since the collision will probably affect the light bulb's behavior, the two perspectives need to interact.

VII. Conclusion

In the previous section, we have stressed the merits of consolidation in comparison to qualitative simulation. To understand what was accomplished, the full context of the discussion must be considered. Three methods of qualitative simulation and our method of consolidation have been summarized. We have argued that consolidation and qualitative simulation solve different information processing tasks. Thus consolidation cannot directly substitute for qualitative simulation, e.g., consolidation cannot predict actual behavior. However, all the methods accept behavioral descriptions in their input, and their output is about behavior, albeit different aspects of behavior. It is possible that this difference is uninteresting, that perhaps wherever consolidation can be used, qualitative simulation can be used to achieve the same effect. Therefore to understand the role of consolidation and the relationships between these tasks, we have shown where consolidation can play a major role in qualitative reasoning: to analyze situations in which the initial conditions and/or outside interactions are not known, to simplify the qualitative simulation of a situation, and to provide a different perspective on causality. A consequence of this argument is that representations of potential behavior should facilitate consolidation in addition to qualitative simulation.

Qualitative reasoning about physical phenonema requires many different kinds of information processing tasks. Qualitative simulation and consolidation do not exhaust the ways that behavior can be inferred. For example, neither process infers the behavior of components from the behavior of situations. Also, there are other information processing tasks such as diagnosis and design for which behavior is not the output. We need to characterize the information that these tasks require, the processes that can perform them, and the representations they use. We also need to integrate these processes and representations so that they can be effectively used to increase the power of problem solving systems.

ACKNOWLEDGMENT

I thank B. Chandrasekaran, John Josephson, Jon Sticklen, and Mike Tanner for their comments.

REFERENCES

[1] D. G. Bobrow, Ed. *Qualitative Reasoning about Physical Systems*. Cambridge, MA: MIT Press, 1985.
[2] P. P. Bonissone, and K. P. Valavanis, "A comparison study of different approaches to qualitative physics theories," in *Proc. Second Conf. Artificial Intell. Appl.*, IEEE Computer Society, Miami Beach, 1985, pp. 236-243.
[3] T. Bylander and B. Chandrasekaran, "Understanding behavior using consolidation," in *Proc. Ninth Int. Joint Conf. Artificial Intell.*, Los Angeles, 1985, pp. 450-454.
[4] T. Bylander, "Consolidation: A method for reasoning about the behavior of devices," Ph.D. dissertation, Lab. for AI Research, CIS Dept., The Ohio State Univ., 1986.
[5] J. de Kleer and J. S. Brown, "The origin, form and logic of qualitative physical laws," in *Proc. Eighth Int. Joint Conf. on Artificial Intell.*, Karlsruhe, 1983, pp. 1158-1169.
[6] ——, "A qualitative physics based on confluences," *Artificial Intell.*, vol. 24, pp. 7-83, 1984.
[7] ——, "Theories of causal ordering," *Artificial Intell.*, vol. 29, pp. 33-61, 1984.
[8] J. L. Dormoy, "Calcul qualitatif (II)," Tech. report 5746-02, Direction des Etudes et Recherches, Electricite de France, 1987.
[9] K. D. Forbus, "Qualitative reasoning about physical processes," in *Proc. Seventh Int. Joint Conf. on Artificial Intell.*, Vancouver, 1981, pp. 326-330.
[10] ——, "Qualitative process theory," *Artificial Intell.*, vol. 24, pp. 85-168, 1984.
[11] ——, "Interpreting observations of physical systems," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-17, no. 3, pp. 350-359, 1987.
[12] D. Gentner, and A. L. Stevens, Eds. *Mental Models*. Hillsdale, NJ: Lawrence Erlbaum, 1983.
[13] J. R. Hobbs and R. C. Moore, Eds. *Formal Theories of the Commonsense World*. Norwood, NJ: Ablex, 1985.
[14] Y. Iwasaki, and H. A. Simon, "Causality in device behavior," *Artificial Intell.*, vol. 29, pp. 3-32, 1986.
[15] B. J. Kuipers, "Commonsense reasoning about causality: Deriving behavior from structure," *Artificial Intell.*, vol. 24, pp. 169-203, 1984.
[16] ——, "Qualitative simulation," *Artificial Intell.*, vol. 29, pp. 289-338, 1986.
[17] ——, "Qualitative simulation as causal explanation," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-17, no. 3, pp. 432-444, 1987.
[18] D. Marr, *Vision*. San Francisco: W. H. Freeman, 1982.
[19] D. S. Weld, "The use of aggregation in causal simulation," *Artificial Intell.*, vol. 30, pp. 1-34, 1986.
[20] B. C. Williams, "Qualitative analysis of MOS circuits," *Artificial Intell.*, vol. 24, pp. 281-346, 1984.

Tom Bylander received the B.S. degree in mathematics and computer science from the University of South Dakota, Vermillion, in 1979, and the M.S. and Ph.D. degrees in computer and information science from The Ohio State University, Columbus, in 1980 and 1986, respectively.

He has stayed at The Ohio State University after completing his degree. He is currently Assistant Professor of computer and information science. His current research interests are in qualitative reasoning about the physical world, knowledge-based reasoning in diagnosis, and explanation-based learning in diagnosis. He is a codesigner of CSRL (Conceptual Structures Representation Language), a language for developing classificatory knowledge-based systems.

Dr. Bylander is a member of the Association for Computing Machinery, the American Association for Artificial Intelligence, the Cognitive Science Society, and the IEEE Computer Society.

88-TB-CORRPRO

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
January, 1988

A CORRECTIVE LEARNING PROCEDURE USING DIFFERENT EXPLANATORY TYPES

Tom Bylander and Michael A. Weintraub
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Note: Accepted to the AAAI Mini -Symposium on Explanation-Based Learning, Stanford University, Palo Alto, CA, March 1988.

A Corrective Learning Procedure Using Different Explanatory Types[1]

Tom Bylander and Michael A. Weintraub
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Corrective learning is the alteration of a system's existing knowledge structures to produce the correct answer when the system's existing structures fail by producing an incorrect response. An explanation-based solution is to compare explanations of why the system produced its incorrect answer with explanations of the correct answer. Explaining the system's answer would be trivial if a single production rule concluded the answer directly from the data. However, the answers from the system we are building will have uncertainty, and credit assignment will involve larger knowledge structures. The problem we are working on is to see how different problem solving structures and underlying models -- and the different types of explanations coming from each -- affect the learning process in the context of corrective learning.

Our work differs from most EBL approaches in the nature of the explanations the system will be producing and using. The usual explanation-based approach is achieved by the construction of a proof showing how an example is an element of some class. The proof can be used to generate a list of sufficient conditions for the identification of some concept. The explanations our work involves can not be construed in the same manner. The answers our system will generate allow for certain conclusions to be inferred from the data, but these conclusions are probabilistic in nature and not definitive. As a result, our system will not produce exact proofs about how some instance belongs to a concept. Instead, our system will only be able to identify a probabilistic relationship between a set of conditions and a concept.

The particular domain we are working in is pathologic gait analysis. Gait analysis is non-trivial. The problem is to properly diagnose which muscles and joints are causing deviations in the gait cycle. For example, patients with cerebral palsy, a disease affecting motor control, typically have several muscles that function improperly in different phases of the gait cycle. The malfunctions in the case of cerebral palsy are improper contractions of the muscles -- both in terms of the magnitude and timing of the muscles -- during the phases of the gait cycle. The problem of diagnosing which muscles and joints are at fault is complicated by interactions between limb segments and attempted compensations by other muscles. In addition, many internal parameters cannot be directly or even indirectly measured using current technology. For example, EMG data is at best a qualitative measure of muscle forces [Simon82].

To perform diagnosis for this kind of problem, our system will consist of structured diagnostic knowledge and a qualitative physical model of human walking. The input to the diagnostic system is the information gathered about a patient by the Gait Analysis Laboratory at the Ohio State University. The data is of three types: clinical, historical, and motion. Clinical data is the result of a physical examination of the patient, and identifies the range of motion of joints by several physical tests. EMG information,

[1]This research is supported by grants 82048-02 from the National Institute for Handicapped Research and 87-0090 from the Air Force Office of Scientific Research.

identifying muscle activity, is also collected. An EMG is not collected on every muscle because of the difficulty involved in attaching electrodes to certain muscle groups. Historical data includes information about any past medical procedures or diagnoses. Motion data identifies the angular position of the patient's joints during the different gait phases. This information is recorded for each plane of interest. The output from the diagnostic system will be an explanation of the malfunctioning gait components so an appropriate therapy can be prescribed. The problem solving structures we are using are based on the theory of generic tasks [Chandra86]. The particular generic tasks involved in the system are abductive assembly, hierarchical classification, and hypothesis matching, respectively used for constructing composite malfunction hypotheses, selecting plausible malfunctions, and combining evidence for and against malfunctions.

In [Chandra87], three types of explanation are identified with knowledge-based systems. These are: (1) trace of run-time, data-dependent, problem solving behavior, (2) understanding the control strategy used by the program in a particular situation, and (3) justifying a piece of knowledge by how it relates to the domain. In our system, the first two types of explanation will be produced by compiled diagnostic knowledge.

To show how the first two explanation types arise, consider the generic task of hierarchical classification. To perform diagnostic reasoning, nodes in a classification hierarchy can be used to represent general and specific malfunctions. During problem solving, the nodes are activated in a top-down fashion and determine their applicability to the current case. Each malfunction that is considered is evaluated by compiled knowledge that matches its features against the data. The confidence value of a malfunction in the classification hierarchy is linked to the data that produced it. This is a type 1 explanation. An example of a type 2 explanation would be to describe why a malfunction was or was not considered. For example, if the confidence value of a general malfunction is low, more specific malfunctions might not be considered.

Type 3 explanations will be produced by the qualitative physical model. These explanations will point out the atypical data that a suspected malfunction would explain, i.e., if the malfunction were true, then the malfunction would be considered the cause of the data. In our system, the qualitative physical model is being implemented by qualitative differential equations [deKleer84, Kuipers86], which will be used to determine how various influences such as muscles and body weight give rise to the observed motion. The model will not be sufficient to identify the correct diagnosis because each part of the observed motion has several possible causes and because of the inherent ambiguity of qualitative models.

The learning in the system will be fault driven, i.e., an incorrect diagnosis is used to focus the learning process. The system, already possessing knowledge about the domain, albeit imperfect, gives an answer to be verified by the domain expert. If the answer is deemed incorrect, the expert provides the "correct" answer. The system must identify how the original answer differs from the correct answer and infer why the expert's answer is better. The system must identify which parts of the problem solving structure caused the incorrect solution, and then modify the structure appropriately.

Explanation of generic task structures (types 1 and 2) will be used to determine which knowledge structures might be at fault. Explanation of the qualitative physical model (type 3) will be compared to the type 1 explanation to select the faulty structure, which might be decomposable into several smaller knowledge structures to be further searched. Once the specific location of an error is found, the type 3 explanation specifies what data should have been used and the other explanation types specify what kinds of adjustments in confidence values will result in preferring the correct answer over the incorrect answer. The following procedure outlines our approach to this problem[2]:

1. Identify initial differences between the system's diagnosis and the correct diagnosis. These differences indicate the sections of the problem solving which need reconsideration. Each difference provides a point from which to focus the learning process. These differences implicate not only the actual compiled knowledge structure that produced the bad judgment, but also the set of decisions leading to the judgment.

2. For each difference, generate explanations of why the system reached its judgment. Specifically, identify the data used in support of the bad judgment (type 1 explanations), and identify the set of decisions leading to the judgment in question (type 2 explanations).

3. Find any commonalities between the explanations of the system's incorrect judgments. Having identified how the incorrect judgment was produced, find any common search strategy or data analysis used in judgments resulting in the set of differences. This step involves comparing the type 2 and 1 explanations produced for each difference, and finding the intersection.

4. Sort the set of commonalities and bad judgments in order of degree of potential effect on correcting the answer if modified, e.g., if a common decison underlies two incorrect judgments, then the changing the common decision may correct both problems.

5. Check consistency. For each element in the set of commonalities and bad judgments, compare the type 3 explanation produced by the qualitative model to the type 1 explanation of the judgment. (The qualitative model does not model the system's control structures, so it does not make sense to include type 2 explanations in this comparison.)

6. Inconsistencies found in the type 1 explanation identify points to correct. Such inconsistencies include: not using all causally relevant information, using data with no causal connection, the sensitivity of some information for decision making is overrated/underrated, etc.

7. Suggest modifications to overcome the inconsistencies. Generate alternatives to the incorrect judgments consistent with the type 3 explanations. This step will focus on making as few changes as possible to correct the overall answer. Each modification includes a proposal of what the type 1 explanation should have been.

8. Select a modification. Choose an acceptable modification based on inconsistencies that were generated.

9. Repeat on underlying knowledge structures. At this point, the chosen modification indicates how a set of judgments and their type 1 explanations should be changed. For each judgment to be changed, the embedded knowledge structures that gave rise to the judgment need to be modified to produce the correct judgment and type 1 explanation.

To illustrate some these steps, consider this oversimplified example. The system chooses hypothesis h_1 with a rating of 8 out of 10 as its answer, and the correct answer rates h_2 with a 6. The question here is to decide how to modify the hypotheses' confidences - whether to increase them or decrease them. The qualitative model will produce an explanation showing how h_1 predicate missed the importance of some data item or possibly overrated itself by overweighting some supporting predicate, or how h_2 might have underrated itself by either underestimating the import of some piece of data or the impact of some predicate. The modification to be selected should result in the rating of h_2 higher than h_1.

[2]Much of this is admittedly vague, because our research is at an early stage.

The learning component will choose a modification from the following choices: including/excluding a predicate to be used in determining confidence in the hypothesis, lower or raise a hypothesis' confidence (or both), or increase/decrease the importance of a hypothesis' predicate. This modification implies that the decisions of underlying knowledge structures need to be changed; thus, these same steps will be applied to them also.

Other work has explored corrective learning using complex knowledge structures. SEEK [Politakis84] and SEEK2 [Ginsberg85], for example, perform corrective learning on structured collections of production rules. Both SEEK and SEEK2 look for statistical properties over a set of cases to discover and modify incorrect rules. This approach assumes that the correct conditions and conclusion for each rule have been identified, but that the logic combining these conditions or the confidence value produced by the rule might not be correct. By adopting an explanation-based approach instead, we intend to provide the capability to alter the conditions in a rule (or larger knowledge structure). Also, an explanation-based approach might lessen the the need for the kind of statistical analysis done by the SEEK programs.

Another example is ACES [Pazzani87], which uses device models for diagnostic reasoning and EBL. ACES uses a mathematical model of the device to confirm or reject fault hypotheses proposed by diagnostic heuristics. Rejected hypotheses cause the modification of diagnostic heuristics based on the reasons the model rejected it. Like ACES, our problem is a diagnostic one, but our system will differ in that our "diagnostic heuristics" will involve more complex pr' 'em solving structures and our device model will be qualitative and will be unable to categorically confirm or reject hypotheses.

Also, both SEEK and ACES assume that only one fault exists. As previously noted, this assumption does not hold in our domain. In fact, a CP patient usually has more than one malfunction.

In this paper, we have outlined our research plan to explore EBL techniques using explanations produced by complex problem solvers. Analysis of pathologic gait is complex because of multiple faults, the interactions between them, and the compensations for them. The analysis itself is the result of a complex problem solving process involving many different problem solving tasks. Several different types of explanations exist, and we plan to investigate how these different explanatory types impact an EBL approach to corrective learning.

References

Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert, 1*(3), 23-30.

Chandrasekaran, B., Tanner, M. C., and Josephson, J. R. (1987). Explanation: The Role of Control Strategies and Deep Models. In Hendler, J. (Ed.), *Expert Systems: The User Interface*. Norwood, New Jersey: Ablex.

de Kleer, J., and Brown, J. S. (1984). A Qualitative Physics Based on Confluences. *Artificial Intelligence, 24*, 7-83.

87-BC-PARADI

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
July 1987

WHAT KIND OF INFORMATION PROCESSING IS INTELLIGENCE? A PERSPECTIVE ON AI PARADIGMS AND A PROPOSAL

B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Note: Section 3 is a minipaper on Connectionism, and can be read separately.

This paper is an extended version of talks that I have given at the AAAI Workshop on Foundations of Artificial Intelligence, Las Cruces, New Mexico, in February, 1986, and at the Workshops on Theoretical Issues in Conceptual Information Processing, Philadelphia, Pennsylvania, in August, 1986, and Washington D.C., June 1987. This paper will appear in *Source Book on the Foundations of AI,* Partridge and Wilks, Editors, Cambridge University Press.

Table of Contents

July 10, 1987

What Kind of Information Processing is Intelligence? A Perspective on AI Paradigms and A Proposal

B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

1. AI as a Science of Intelligence

Theoretical and empirical work in artificial intelligence (AI) has now gone on for close to thirty years, but few minds have been changed about most of the central philosophical issues surrounding mind. People who asserted in the early days of AI that machines cannot think still assert that machines cannot think. People who believed that machines cannot be conscious or feel pain still believe that machines cannot be conscious or feel pain. Even with respect to the idea that computation over discrete symbol systems, the so-called "symbolic paradigm" (Smolensky, 1988), is a pretty good basis for capturing intelligence -- by far the dominant paradigm in AI and the reason AI is closely associated with computer science -- there are recurring doubts. In addition to the people out there who have always felt we need holograms or chemistry for a proper account of intelligence, we now have a connectionist school that rejects the symbolic paradigm for intelligence. In spite of all this, work in AI has steadily attracted its share of philosophers and psychologists -- not to speak of people who see the commercial possibilities of mechanized intelligence -- who feel that AI is exciting and important both in its overall view of intelligence as well as in some of its concrete achievements.

Minds and Intelligences. Let us make a useful distinction which might eliminate at least some of the arguments about AI: the distinction between "intelligence" and "mind." Many discussions on the philosophical implications of AI-- e.g., the numerous articles of the 50's and 60's on minds and machines-- equated the question, "Can machines be intelligent?" with "Are minds machines?". There is a useful alternative to this equation of mind and intelligence, viz., that intelligence is a *tool* of the mind. In fact there is a tradition in Hindu and Buddhist philosophies which embodies precisely such a distinction: it views intelligence as an internal sense organ much as sight is an external sense organ. As a sense organ, it interprets the world and makes the information available to the "watcher." My aim in making this distinction here is not to stake an ultimate position about the irreducibility of mind to mechanism, but merely to remove from discussion some elements about which AI as a technical discipline has nothing to say at this time. Even the most rabid mechanist within AI will need to admit that while AI may have impressively useful things to say about cognition and perception, it simply has nothing technical -- at this point -- to say about consciousness, feelings, will, etc. Thus from a technical viewpoint, I want to take intelligence, not mind, as the current subject matter of AI. If we have succeeded in taking vitalism out of our limbs and cells, we hope similarly to take mysticism at least out of intelligence.

Paradigmatic confusion in AI. In spite of what I regard as significant achievements of AI in beginning to provide a computational language to talk about the nature of intelligence, the not so well-kept secret is that AI is internally in a paradigmatic mess. There is really no broad agreement on the essential nature or formal basis of intelligence and the proper theoretical framework for it.

1.1. Intelligence as Information Processing on Representations

However, let us first seek some unities. There *is* something that is shared *almost universally* among workers in AI: "Significant (all?) aspects of cognition and perception are best understood/modeled as *information processing activities on representations.*" The dominant tradition within AI has been the symbolic paradigm[1]. On the other hand, modern connectionists (and the earlier perceptron theorists) offer largely analog processes implemented by weights of connections in a network. Stronger versions of the symbolic paradigm have been proposed by Newell as the physical symbol system hypothesis (Newell, 1980) and elaborated by Pylyshyn(1984) in his thesis that computation is not simply a metaphorical language to talk about cognition, but that cognition is literally computation over symbol systems. It is important to emphasize that this thesis does not imply a belief in the practical sufficiency of current von Neuman computers for the task, or a restriction to serial computation. Often disagreements with the symbolic paradigm turn out to be arguments for paralell computers of some type rather than arguments against computations on discrete symbolic representations.

The above description of intelligence as information processing does not, however, characterize the class of intelligent processes well enough within the class of all information processing transformations. Is there something that can be recognized as the *essential* nature of intelligence that can be used to characterize all its manifestations: human, alpha-centaurian and artificial? It is possible that intelligence is merely a somewhat random collection of information processing transformations acquired over eons of evolution, but in that case there can hardly be an interesting science of it. It is also possible that there need not be anything that particularly restricts attempts to make intelligent machines, i.e., while there may well be characterizations of human intellectual processes, they need not be taken to apply to other forms of intelligence. While in some sense this seems right -- human intellectual processes do not bound the possibilities for intelligence -- nevertheless I believe that there is an internal conceptual coherence to the class of information processing activities characterising intelligence. The oft-stated dichotomy between the simulation of human cognition versus making machines smart is a temporarily useful distinction, but its implication that we are talking about two very different phenomena is, I believe, incorrect. In any case, a task of AI as a science is to explain human intelligence. The underlying unity that we are seeking can be further characterised by asking, "What is it that unites Einstein, the man on the street in a western culture, and a tribesman in a primitive culture, as information processing agents?"

In this paper, my aim is to give a broad brush treatment of the attempts to understand the nature of intelligence. By their very nature, such broad brush accounts tend to treat history a bit too neatly. Another consequence of this is that an approach might be treated as belonging to a certain class, while the authors of the approach in question might not share the implications of the classification. But a treatment in such broad terms is nevertheless necessary to make sense of a field such as AI which is in some degree of conceptual confusion about its foundations.

[1]I am unhappy with this term to describe the commitment to computation over discrete symbolic systems, since I think that all representations are symbolic, otherwise they wouldn't be representations. There is realy no satisfactory, generally agreed brief term for this. Fodor and Pylyshyn (1988) use the term "classical" models. Dennett (1986) uses the term "High Church Computationalism" and so on. I will stick with the terms "symbolic paradigm," "symbolic approaches," and "symbolic computationalism" to refer to computation over discrete symbol systems.

My aim is not to give a quick tutorial on the history of AI, or even the various technical ideas in AI. I am only concerned with what AI has had to say about the question, "What kind of information processing is intelligence?" Also, this paper is really written for the AI researcher who already knows the various theories. What I plan to do is offer a view of how the field really works as a discipline, and how some disagreements can be understood only by tracing them to the root of the problem: disagreements about the nature of the science.

2. AI Theories from the 40's to the 60's

2.1. Pre- and Quasi-Representational Theories

Let us now trace the various streams in AI that attempted to come to grips with the nature of intelligence. The period under survey can be characterized as a transition from formalisms with an essentially non-representational character through ideas which oscillated between brain-level vs mind-level representations finally to a clear dominance of discrete symbolic representations within AI and emphasis on higher cognitive phenomena.

The earliest of the modern attempts in this direction was the *cybernetics* stream, associated with the work of Wiener (1948) who laid some of the foundations of modern feedback control[2]. The importance of cybernetics was that it suggested that *teleology could be consistent with mechanism.* The hallmark of intelligence was said to be *adaptation*, and since cybernetics seemed to provide an answer to how this adaptation could be accounted for with feedback of *information.* and also account for teleology (e.g., "The *purpose* of the governor is to keep the steam engine speed constant"), it was a great source of early excitement for people attempting to model biological information processing. However, Cybernetics never really became the language of AI, because it did not have the richness of ontology to talk about cognition and perception: while it had the notion of information processing in some sense, i.e., it had goals and mechanisms to achieve them, it lacked the notion of computation, not to mention representations.

Modeling the brain as automata (in the sense of automata theory) was another attempt to provide a mathematical foundation for intelligence. For example, the finite automata model of nervenets that McCulloch and Pitts (1943) proposed was among the first concrete postulations about the brain as a computational mechanism. Automata models were computational, i.e., they had states and state transition functions, and the general theory dealt with what kinds of automata can do what kinds of things. While this was a source of great excitement -- one should try to imagine being present at the time when the computer, information theory and the automata theories were all being born at about the same time, and the sense of exhilaration that must have resulted from the thought that a formal language in which to talk about minds and brains was within reach! -- in retrospect, automata theory didn't have enough of the right kind of primitive objects for talking about the phenomena of cognition and perception. What AI needed was not theories *about* computation but theories which were descriptions of particular computations, i.e., really programs that embody theories of cognition. Naturally enough, automata theory evolved into the formal foundation for some aspects of computer science, but its role in AI *per se* tapered off.

Another strain, which was much more explicit in its commitment to seeking intelligence by

[2]Cybernetics as a movement had broader concerns than the issues surrounding feedback control, as applied by Wiener to understanding control and communication in animals and machies. Information and automata theories and neural nets as automata were all part of the cybernetic milieu of bringing certain biological phenomena under the rigor of formalisms. I discuss automata and neural nets shortly.

modeling its seat, the brain, looked at neurons and neural networks as the units of information processing out of which thought and intelligence can be explained and produced. Neural net simulation and the work on Perceptrons (Rosenblatt, 1962) are two major examples of this class of work. Its lineage can be traced to Hebb's work (Hebb, 1949) on cell assemblies which had a strong effect on psychological theorizing. Hebb proposed a dynamic model of how neural structures could sustain thought, how simple learning mechanisms at the neural level could be the agents of higher level learning at the level of thought. In retrospect, there were really two rather distinct kinds of aims that this line of work pursued. In one, an attempt was made to account for the information processing of neurons and collections of them. To the extent that it is generally granted that neural structures form the implementation medium of human intelligence and thought, this seems like an eminently important line of investigation. In fact, over the years, concrete identifications have been made of particular functions computed by particular neural structures in the brain, and these data may eventually form the empirical basis of any theory of how brains and minds can be bridged analytically.

In the other line of work in neural models – prefiguring the claims of latter day connectionism –– the attempt is to explain intelligence directly in terms of neural computations. Since in AI explanation of intelligence takes the form of constructing artifacts which are intelligent, this is a tall order –– the burden of producing programs which simulate neural–like mechanisms on one hand, and at the same time do what intelligent agents do: solve problems, perceive, explain the world, speak in a natural language, etc. is a heavy one. There is a problem with the level of description here –– the terms of neural computation seem far removed from the complex content of thought –– and bridging it without hypothesizing levels of abstraction between neuronal information processing and highly symbolic forms of thought is difficult. In other words, even if it is true that the brain is made up completely of neural structures of certain types whose behavior is fully understood, and if one is given a bucketful of such neural strcutures one would still be not very close to constructing a natural language understanding program without theories of knowledge and syntax and semantics. The general temptation in this area has been to sidestep the difficulties by assuming that appropriate learning mechanisms at the neural level can result in sufficiently complex high level intelligence, much as it presumably occurred in evolution, so that the designer of the artifact need not have theories of cognition or perception at levels higher than the neural level. But the difficulty of getting the necessary learning to take place in less than evolutionary time has generally resulted in the neural network level not being a serious contender for AI theory making and system construction until a new generation of connectionist models began to admit representations of high level abstractions directly. Because it raises not only the level of abstraction issues but also issues about the nature of representations, I propose to discuss connectionism separately (Section 3).

A large body of work, mainly statistical in character, developed under the rubric of pattern recognition (see a text such as (Duda, 1973)). It identified the problem of recognition with classification and developed a number of statistical algorithms for classification. While it had a number of representational elements (the object in question was represented as a vector in a multidimentional space) and shared some of the concerns with the perceptron work (linear or nonlinear separability of patterns in N–dimensional spaces), it developed into a mathematical discipline of its own without making a significant impact on the overall concerns of AI. In (Chandrasekaran, 1986a), I discuss how more flexible representations are increasingly needed for even the classification problem as the complexity of the problem increases.

A number of reasons can be cited for the failure of all this class of work, viz., perceptrons, neural nets, and statistical classification to hold center stage in AI. The loss of interest in perceptrons is often attributed to the demonstration by Minsky and Papert (1969) of their inadequacies. I believe, however, that their demonstration was in fact limited to single layer perceptrons, and was not the real reason for their disappearance from the scene. The real reason, I believe, is that powerful representational and representation manipulation tools were missing. The alternative of discrete symbolic representations quickly filled this need, and provided an experimental medium of great flexibililty.

2.2. Early AI Work Based on Symbolic Representations

The final transition to discrete symbolic representations was rather quick. The mathematics of computabilty also made some investigations along this line attractive and productive. The end of the period saw not only a decisive shift towards representational approaches, but the particular kind of representationalism that became the common currency was the symbolic paradigm.

Early work in this computationalist spirit took on two major forms:

1. Show mathematically that certain functions thought to be characteristic of intelligence were *computable*, (e.g., induction machines of Solomonoff (1957), Gold's work on learning of grammars (Gold, 1967)). From the viewpoint of *constructing* artifacts that would perform these functions these results would in fact have been depressing if notions of *complexity of computation*, which were later to be developed in computer science with great elegance and precision, had been available at that time. The algorithms proposed were typically computationally intractable. However, in reality, this family of results was important to AI in making the idea of machine intelligence theoretically more plausible by showing that the mental functions as defined by reasonable appeal to intuition were in fact computable -- a certain amount mysticism that would otherwise surround terms such as "induction" and "learning" was thus eliminated[3].

2. Demonstrate the possibility of AI by building computer programs that solve problems requiring intelligence. Game playing programs, the Analogy program of Evans, the scene analysis program of Guzman and the Logic Theorist and the heuristic compiler of Newell and Simon, etc., etc., showed that the underlying features of intelligence of which they were meant to be demonstrations were in fact capable of artifactual embodiment. Of course the intent of these programs was not merely that -- each of which was also an exploration of a general theory of some sort, problem solving, scene analysis, analogical reasoning, etc.-- but not all of them led to more general theories about fundamental issues in AI. But, whatever their original intent, all these early programs ended up playing the roles of "realities in the field" to counteract the intellectual milieu of that time which resisted the no'ion of intelligence as mechanical.

In general, the net result of most (not all) of this work was socio-psychological: it made the idea of AI plausible, and blunted the first round objections, such as , "Ah, but machines cannot learn," and "Ah, but machines cannot create." These early programs were also the means by which psychologists and philosophers became aware of the new kid on *their* block. The attention that AI gained at that time has continued to this day.

3. On the Nature of Representations: Connectionism vs The Symbolic Paradigm

Let me restate some of the terminology here. I have called the hypothesis that intelligence can be

[3] It is an open question whether our understanding of induction, e.g., was in fact genuinely advanced from the viewpoint of building machines to perform them.

accounted for by algorithmic processes which interpret discrete symbol systems the *symbolic paradigm* or *symbolic approaches*. Let us call the alternative to this the *non-symbolic paradigm or approaches*, for lack of a better word. Connectionism is an example of this alternative, though not the only one.

A source of confusion is that connectionist theorists often use algorithmic language for describing parts of their systems' behavior. It is my belief (see Section 3.3) that such an algorithmic specification is quite irelevant, and does not involve basic representational commitments. I want to reserve the term "symbolic" approaches to those theories which make representational commitments at the theoretical level for discrete symbol systems.

Since connectionism challenges some of the basic assumptions under which much AI work has gone on for the past several decades, it is important to spend some time examining the nature of representations, and the differences between the symbolic paradigm and connectionism in this regard. This section will be in the form of a detour, since the major part of this paper (from Sections 4 onward) will be on the theories of the past two decades in the symbolic paradigm.

3.1. The Roots of the Debate

The connectionism– symbolic computationalism debate in AI today is but the latest version of a fairly classic contention between two sets of intuitions each leading to a *weltanschauung* about how to study the phenomena of current interest. The debate can be traced at least as far back as Descartes in modern times (and to Plato if one wants to go further back) and the mind–brain dualism that goes by the name of Cartesianism. In the Cartesian worldview, the phenomena of mind are exemplified by language and thought. These phenomena may be implemented by the brain, but are seen to have a constituent structure in their own terms and can be studied abstractly. Logic and symbolic representations have often been advanced as the appropriate tools for studying these phenomena.

Functionalism in philosophy, information processing theories in psychology, and the symbolic paradigm in AI all share these assumptions. While most of the intuitions that drive this point of view arise from a study of cognitive phenomena, the thesis is often extended to include perception, as e.g. in Bruner's thesis (Bruner, 1957) that *perception is inference*. In its modern versions this viewpoint appeals to Turing's Hypothesis as providing a justification for limiting attention to symbolic computational models. These models ought to suffice, the argument goes, since even continuous functions can be computed to arbitrary precision by a Turing machine.

The opposition to this view springs from anti–Cartesian intuitions. My reading of the philosophical impulse behind anti–Cartesianism is that it is a reluctance to assign any kind of ontological independence to mind, a reluctance arising from the feeling that mind–talk is but an invitation to all kinds of further mysticisms, such as soul–talk. Thus anti–Cartesians tend to be materialists with a vengeance, and are skeptical of the separation of the mental from the brain–level phenomena. Additionally, the brain is seen to be nothing like the symbolic processor needed to support the symbolic paradigm. Instead of what is seen as the sequential and combinational perspective of the symbolic paradigm, some of the theories in this school embrace parallel, "holistic", non–symbol–processing alternatives, while others do not even subscribe to any kind of information processing or representational language in talking about mental phenomena. Those who think information processing of some type is still needed nevertheless reject processing of labeled symbols, and look to analog or continuous processes as the natural medium for modeling the relevant phenomena. In contrast to Cartesian theories, most of the concrete work in these schools deals with perceptual (or even motor) phenomena, but the framework is meant to cover complex cognitive phenomena as well. Eliminative materialism in philosophy, Gibsonian theories in psychology, connectionism in psychology and AI, all these can be grouped as more or less sharing this perspective, even though they differ among each other in a number of issues. For example, the Gibsonian direct perception theory is anti–representational. Perception, in this view, is not inference nor a product of any kind of information processing, but is a one–step mapping from stimuli to categories of perception, made possible by the inherent properties of the perceptual architecture. All the needed distinctions are already there directly in the architecture, and no processing over representations is needed. To put it simply, the brain is all there is and it isn't a computer either.

Note that the proponents of the symbolic paradigm can be happy with the proposition that mental phenomena are implemented by the brain, which may or may not itself have a computationalist ac– count. However, the anti–Cartesian cannot accept this duality. He is out to show the mind as epiphenomenal.

I need to caution the reader that each of these positions that I have described above is really a composite. Few people in either camp subscribe to all the features in my description of them. Most of them may not even be aware of themselves as participating in such a classic debate. In par– ticular, many connectionists may bristle at my inclusion of them on the side of the debate that I did, since their accounts are laced with talk of "connectionist inference" and algorithms for the units. The algorithmic accounts in my view are incidental. (I discuss this further in Section 3. 3.) But my account, painted with a broad brush as it is, is helpful to understand the rather diverse collection of bedfellows that connectionism has attracted.

Connectionism is a recent and less radical member of this camp. Many connectionists do not have a commitment to brain–level theory making. It is also explicitly representational, its only argument being about the medium of representation.

I believe that there is in fact a great deal of unanalyzed assumptional baggage in each of these classes of theories. I will try to show that connectionism is a corrective to some of the basic as– sumptions in the symbolic paradigm, but for most of the central issues of intelligence, connectionism is only marginally relevant.

As a preliminary to the discussion, I want to try to pin down, in the next subsection, some es– sential distinctions between the symbolic and nonsymbolic approaches to information processing.

3.2. Symbolic and Non-Symbolic Representations

Consider the problem of multiplying two integers. We are all familiar with algorithms to perform this task. We also know how the traditional slide rule can be used to do this multiplication. The multiplicands are represented by their logarithms on a linear scale, which are then "added" by being set next to each other, and the result is obtained by reading off the sum's anti–logarithm. While both the algorithmic and slide rule solutions are *representational*, in no sense can either of them be thought of as an "implementation" of the other. They make very different commitments about what is represented. There are also striking differences between them in practical terms. As the size of the multiplicands increases, the algorithmic solution suffers in the amount of *time* it takes to complete the solution, while the slide–rule solution suffers in the amount of *precision* it can deliver.

Let us call the algorithmic and slide–rule solutions C1 and C2. There is yet another solution C3, which is the simulation of C2 by an algorithm. C3 can simulate C2 to any desired accuracy. But C3 has radically different properties from C1 in terms of the information that it represents. C3 is closer to C2 representationally. Its symbol manipulation character is at a lower level of abstraction altogether. Given a blackbox multiplier, ascription of C1 or C2 (among others) as what is *really* going on makes for different theories about the process. Each theory makes different ontological commitments. Further, while C2 is "analog" or continuous, the existence of C3 implies that the essential characteristic of C2 is not continuity *per se*, but a radically different sense of representation and processing than C1.

An adequate discussion of what makes a symbol in the sense used in computation over symbol systems requires a much larger space and time than I have at present (see (Pylyshyn, 1984) for a thorough and illuminating discussion of this topic), but the following points seem useful. There is a type–token distinction that seems relevant: symbols are types about which abstract rules of behavior are known and can be brought into play. This leads to symbols being labels which are "interpreted" during the process, while there are no such interpretations in the process of slide rule multiplication (except for input and output). The symbol system can thus represent *abstract forms*, while C2 above performs its addition or multiplication not by instantiating an abstract form, but by having, in some sense, *all* the additions and multiplications directly in its architecture.

While I keep using the word "process" to describe both C1 and C2, strictly speaking there is no process in the sense of a temporally evolving behavior in C2. The architecture directly produces the solution. This is the intuition behind the Gibsonian direct perception in contrast to the Bruner alternative of perception as inference[4]: the process of inference implies a temporal sequentiality. Connectionist theories have a temporal evolution, but at each cycle, the information process does not have a step–by–step character like algorithms do. Thus the alternatives in the non–symbolic paradigm are generally presented as "holistic."

The main point of this section is that there exists functions for which symbol and non–symbol system accounts differ fundamentally in terms of representational commitments.

3.3. Connectionism and Its Main Features

While connectionism as an AI theory comes in many different forms, they all seem share to the idea that the *representation* of information is in the form of *weights* of connections between processing units in a network, and information processing consists of (i) the units transforming their input into some output, which is then (ii) modulated by the weights of connections as inputs to other units. Connectionist theories especially emphasise a form of learning which is largely in the form of continuous functions adjusting the weights in the network. In some connectionist theories the above "pure" form is mixed with symbol manipulation processes. My description is based on the abstraction of connectionist architectures as described by Smolensky (1988). Smolensky's description captures the essential aspects of the connectionist architecture.

A few additional comments on what constitutes the essential aspects of connectionism may be useful, especially since connectionist theories come in so many forms. My description above is couched in non–algorithmic terms. In fact many connectionist theorists describe the units in their systems in terms of algorithms which map their inputs into discrete states. My view is that the discrete state description of the units' output as well as the algorithmic specification of the units' behavior is not substantially relevant. Smolensky's statement that differential equations are the appropriate language to use to describe the behavior of connectionist systems lends credence to my summary of connectionist systems.

While my description is couched in the form of continuous functions, the arguments in the Section

[4]Whether perception, if it is an inferential process, necessarily has to be continuous with cognitive processes, i.e., they all have access to one knowledge base of an agent is a completely different issue (Fodor, 1983). I am mentioning it here because the perception as inference thesis does not necessarily mean one monolithic process for all the phenomena of intelligence.

3.2 indicate that it is not in the property of continuity per se that the essential aspect of the architecture lies, but in the fact that the connectionist medium has no internal labels which are interpreted and no abstract forms which are instantiated during processing. Thus connectionist models stand in the same relationship to the symbolic models that C2 does to C1 in my discussion in Section 3.2.

There are a number of properties of such connectionist networks that are worthy of note and that explain why connectionism is viewed as an alternative paradigm to the symbolic theories.

- Parallelism: While theories in the symbolic paradigm are not restricted to serial algorithms, connectionist models are intrinsically parallel, in most implementations massively parallel.

- Distributedness: Representation of information is *distributed* over the network in a very specialized sense. i.e., the state vector of the weights in the network *is* the representation. The two properties of parallelism and distribution have attracted adherents who feel that human memory has a "holistic" character -- much like a hologram -- and consequently have reacted negatively to discrete symbol processing theories, since these compute the needed information from parts and their relations. Dreyfus (1979), e.g., has argued that human recognition does not proceed by combining evidence about constituent features of the pattern, but rather uses a holistic process. Thus Dreyfus looks to connectionism as vindication of his long-standing criticism of AI. Connectionism is said to perform "direct" recognition, while symbolic AI performs recognition by sequentially computing intermediate representations.

- Softness of constraints (Smolensky, 1988): Because of the continuous space over which the weights take values, the behavior of the network, while not necessarily unimodal, tends to be more or less smooth over the input space.

The above characteristics are especially attractive to those who believe that AI must be based more on brain-like architectures, even though within the connectionist camp there is a wide divergence about the degree to which directly modeling the brain is considered appropriate. While some of the theories explicitly attempt to produce neural-level computational structures, some others (see e.g, (Smolensky, 1988)) propose a "subsymbolic level" intermediate between symbolic and neural level theories, and yet others offer connectionism as a computational method that operates in the symbolic level representation itself. The essential idea uniting them all is that the totality of connections defines the information content, rather than representing information as a symbol structure.

3.4. Is Connectionism Merely An Implementation Theory?

Two kinds of arguments have been made that connectionism can at best provide possible implementations for algorithmic AI theories. The traditional one, viz., that symbolic computationalism is adequate, takes a couple of forms. In one, continuous functions are thought to be the alternative, and the fact that they can be approximated to an arbitrary degree of approximation is used to argue that one need only consider algorithmic solutions. In the other, connectionist architectures are thought to be the implementation medium for symbolic theories, much as the computer hardware is the implementation medium for software. In an Section 3.2, I have considered and rejected these arguments. I showed that in principle the symbolic and non-symbolic solutions may be alternative theories in the sense that they may make different representational commitments.

The other argument is based on a consideration of the properties of high level thought, in particular language and problem solving behavior. Connectionism by itself does not have the constructs, the argument runs, for capturing these properties, so at best it can only be a way to implement the higher level functions. I will discuss this and related points in Section 3.8.

Having granted that connectionism (actually, non–symbolic theories in general) can make a theoretical difference, I now want to argue that the difference connectionism makes is relatively small to the practice of most of AI. This is the task of the rest of Section 3.

3.5. Need for Compositionality

Proponents of connectionism sometimes claim that solutions in the symbolic paradigm are composed from constituents, while connectionist solutions are holistic, i.e., they cannot be explained as compositions of parts. Composition, in this argument, is taken to be intrinsically an algorithmic process.

Certainly, for some simple problems there exist connectionist solutions with this holistic character. For example, there are connectionist solutions to character recognition which directly map from pixels to characters and which cannot be explained as composing evidence about the features such as closed curves, lines and their relations. Character recognition by template matching, though not a connectionist solution, is another example whose information processing cannot be explained as feature composition. But as problems get more complex, the advantages of modularization and composition are as important for connectionist approaches as they are for house–building or algorithmic AI. A key point is that composition may be done connectionistically, i.e., it does not always require algorithmic methods.

To see this, let us consider word recognition, a problem area which has attracted significant connectionist attention (McClelland, Rumelhart, and Hinton, 1986). Let us take the word "QUEEN"[5] A "featureless" connectionist solution similar to the one for individual characters can be imagined, but a more natural one would be one which in some sense composes the evidence about individual characters into a recognition of the word. In fact, the connectionist solution in (McClelland, et al, 1986) has a natural interpretation in these terms. The fact that the word recognition is done by composition does not mean either that each of the characters is explicitly recognized as part of the procedure, or that the evidence is added together in a step by step, temporal sequence.

Why is such a compositional solution more natural? Reusability of parts, reduction in learning complexity as well as greater robustness due to intermediate evidence are the major computational advantages of modularization. If the reader doesn't see the power of modularization for word recognition, he can consider sentence recognition and see that if one were to go directly from pixels to sentences without in some sense going through words the number of recognizers and their complexity would have to be very large even for sentences of bounded length.

To use another example, if one has a system that already recognizes "Monkey," "banana," and "Eat(a, b)", then recognizing "Monkey eats banana," without composing the constituent recognizing capabilities above would be very wasteful of resources and would require excessive learning times as well. Composition is a powerful aid against complexity whether the underlying system is connectionist or algorithmic. Of course, connectionism provides one style for composition and algorithmic methods another, each with its own "signature" in terms of the details of performance.

These examples also raise questions about the claims of distributedness of connectionist representations. For complex tasks, information is in fact localized into portions of the network. Again, in McClelland, et al's network for word recognition physically local subnets can be identified, each corresponding to one of the characters. Thus the the hopes of some proponents for almost holographic distributedness of representation are bound to be unrealistic.

[5]My description of word recognition is modeled after the example given in (McClelland, et al) cited above. The word that they discuss is "TAKE".

3.6. *Information Processing Level Abstractions*

Marr (1982) originated the method of information processing analysis as a way of separating the essential elements of a theory from implementation level commitments. He proposed that the following methodology be adopted for this purpose. First, identify an information processing function with a clear specification about what kind of information is available for the function as input and what kind of information needs to be made available as output by the function. Then specify a particular information processing (IP) theory for achieving this function by stating what kinds of information the theory proposes need to be represented at various stages in the processing. Actual algorithms can then be proposed to carry out the IP theory. These algorithms will make additional representational commitments. For example, he specified that one of the functions of vision is to take as input image intensities in a retinal image, and produce as output a 3-dimensional shape description of the objects in the scene. His theory of how this function is achieved in the visual system is that three distinct kinds of information need to be generated: from the image intensities, a primal sketch of significant intensity changes -- a kind of edge description of the scene -- is generated, then a description of surfaces of the objects and their orientation, what he called a 2 1/2 -dimensional sketch is produced from the primal sketch, and finally a 3-d shape description is generated.

Even though Marr talked the language of algorithms as the way to realize the IP theory, there is in principle no reason why portions of the implementation cannot be done connectionistically.

Thus IP level abstractions constitute the top level content of much AI theory making. In the example about recognition of the word "QUEEN" in Section 3.5, the IP level abstractions in terms of which the theory of word recognition was couched were the evidences about the presence of individual characters. The difference between schemes in the symbolic and connectionist paradigms is that these evidences are labeled symbols in the former, which permit abstract rules of compositions to be invoked and instantiated, while in the latter they are represented more directly and affect the processing without undergoing any interpretive process. Interpretation of a piece of a network as evidence about a character is a design and explanatory stance, and is not part of the actual information processing.

As connectionist structures evolve (or are built) to handle increasingly complex phenomena, they will end up having to incorporate their own versions of *modularity* and *composition*. Already we saw this in the only moderately complex word recognition example. When and if we finally have connectionist implementations solving a variety of high level cognitive problems (say natural language understanding or planning or diagnosis), the design of such systems will have an enormous amount in common with the corresponding symbolic theories. This commonness will be at the level of information processing abstractions that both classes of theories would need to embody. In fact, the *content* contributions of many of the nominally symbolic theories in AI are really at the level of the IP abstractions to which they make a commitment, and not to the fact that they were implemented in a symbolic structure. Symbols have often merely stood in for *abstractions* that need to be captured one way or another, and have often been used as such. The hard work of theory making in AI will always remain at the level of proposing the right IP level of abstractions, since they provide the content of the representations. The decisions about which of the transformations are best done by means of connectionist networks, and which using symbolic algorithms, can properly follow once the IP level specification of the theory has been given.

Thus, connectionist (and symbolic) approaches are both *realizations* of a more abstract level of description, viz., the *information processing (IP) level.*[6]

Rumelhart and McClelland (1986) comment that symbolic theories that are common in AI are really explanatory approximations of a theory which is connectionist at a deeper level. To take the "QUEEN" example again, saying that the word is recognized by combining evidences about individual characters in a certain way may appear to be giving an algorithmic account, but this description is really neutral regarding whether the combination is to be done connectionistically or algorithmically. It is not that connectionist structures are the reality and symbolic accounts provide an explanation, it is that it is the IP abstractions contained in AI theories that contain a large portion of the explanatory power.

I argued, in Section 3.2, that given a function, the approaches in the symbolic and non-symbolic paradigms may make rather different representational commitments; in compositional terms, they may be composing rather different subfunctions. In this section I am arguing, seemingly paradoxically, that for complex functions the two theories converge in their representational commitments. A way to clarify this is to think of two stages in the decomposition: an architecture-independent and an architecture-dependent one. The former is an IP theory that will be realized by particular architectures for which additional decompositions will need to be made. Simple functions such as multiplication are so close to the architecture level that we only saw the differences between the representational commitments of the algorithmic and slide rule solutions. The word recognition problem is sufficiently removed from the architectural level that we saw macrosimilarities between computationalist and connectionist solutions. The final performance will of course have microfeatures that are characteristic of the architecture (such as the "softness of constraints" for connectionist architectures).

Where the architecture-independent theory stops and the architecture-dependent starts does not have a clear line of demarcation. It is an empirical issue, partly related to the primitive functions that can be computed in a particular architecture. The farther away a problem is from the architectures' primitive functions, the more architecture-independent decomposition needs to be done at design time. I believe that certain kinds of retrieval and matching operations, and parameter learning by searching in local regions of space are especially appropriate primitive operations for connectionist architectures.

3.7. Learning to the Rescue?

What if connectionism can provide learning mechanisms such that one starts without any such abstractions represented, and the system learns to perform the task in a reasonable amount of time? In that case, connectionism can sidestep pretty much all the representational problems and dismiss them as the bane of the symbolic paradigm. The fundamental problem of complex learning is the *credit assignment problem*, i.e, the problem of deciding what part of the system is responsible for either the correct or the incorrect performance in a case, so that the learner knows how to change the structure of the system. Abstractly, the range of variation of the structure of a system can be represented as a multi-dimensional space of parameters, and the process of learning as a search process in that space for a region that corresponds to the right structure of the systems. The more complex the system, the vaster the space in which to do the search. Thus learning the correct set of parameters by search methods which do not have a powerful notion of credit assignment would work in small search spaces, but would be computationally prohibitive for realistic problems. Does connectionism have a solution to this problem?

[6] I am indebted to Dean Allemang and Ashok Goel who noted that at this level of abstraction the distinctions between the two approaches to complex problems become small relative to their commonalities.

If one looks at particular connectionist schemes that have been proposed for some tasks such as learning tense endings (Rumelhart and McClelland, 1986b), a significant part of the abstractions needed are built into the architecture in the choice of inputs, feedback directions, allocation of sub-networks, and the semantics that underlie the choice of layers for the connectionist schemes. That is, the inputs and the initial configuration incorporate a sufficiently large part of the abstractions needed that what is left to be discovered by the learning algorithms, while nontrivial, is proportionately small. The initial configuration decomposes the search space for learning in such a way that the search problem is much smaller in size. In fact the space is sufficiently small that statistical associations can do the trick.

The recognition scheme for "QUEEN" again provides a good example for illustrating this point. In the McClelland, et al, scheme that I cited earlier essentially the decisions about which subnet is going to be largely responsible for "Q", which for "U," etc, as well as how the feedback is going to be directed are all made by the experimenter before learning starts. The underlying IP theory is that evidence about individual characters is going to be formed directly from the pixel level, but recognition of "QU" will be done by combining information about the presence of "Q" and "U", as well as their joint likelihood. The degree to which the evidence about them will be combined is determined by the learning algorithm and the examples. In setting up the initial configuration, the designer is actually programming the architecture to reflect the above IP theory of recognizing the word. An alternate theory for word recognition, say one that is more holistic than the above theory (i.e., one that learns the entire word directly from the pixels) will have a different initial configuration. (Of course, because of lack of guidance from the architecture about localizing search during learning, such a network will take a much longer time to learn the word. But that is the point: the designer recognized this and set up the configuration so that learning can occur in a reasonable time.) Thus while the connectionist scheme for word recognition still makes the useful *performance* point about connectionist architectures for problems that have been assumed to require a symbolic implementation, a significant part of the leverage still comes from the IP abstractions that the designer started out with, or have been made possible by an earlier learning phase working with highly structured configurations.

Additionally, the system that results after learning has a natural interpretation in terms of the abstractions that are needed to solve the problem; the learning process can be interpreted as having successfully searched the space for those additional abstractions that are needed to solve the problem. Thus, connectionism is one way to map from one set of abstractions to a more structured set of abstractions. Most of the representational issues remain, whether or not one adopts connectionism for such mappings.

Of course in human learning, while some of the abstractions needed are "programmed" in at various times through explicit instruction, a large amount of learning takes place without any "designer" intervention in setting up the learning structure as I described in the "QUEEN" example. But there is no reason to believe that humans start with a structure- and abstraction-free initial configuration. In fact, in order to account for the power of human learning, the initial configurations that a child starts out with will need to contain complex and intricate representations sufficient to support the learning process in a computationally efficient way.

3.8. The Domains for Connectionism and Symbolic Computations

For this discussion, a distinction between "micro" and "macro" phenomena of intelligence is useful. Rumelhart, McClelland, et al (1986) use the former term in the subtitle of their book to indicate that the connectionist theories that they are concerned with deal with the fine details of processes. A duration of 50-100 milliseconds has often been suggested as the size of the temporal "grain" for processes at the micro level. Macro phenomena take place over seconds if not minutes in the case of a human. These evolve over time in such a way that there is a clear temporal ordering of some of its major behavioral states. For example, take the problem solving behavior represented by the GPS problem solver. The agent is seen to have a goal at a certain instant. to set up a subgoal at another instant, and so on. Within this problem solving behavior, the selection of an appropriate operator, which is typically modeled in GPS implementations as a retrieval algorithm from a Table of Connection, could be a "micro" behavior. Many of the phenomena of language and reasoning have a large macro component. Thus this domain includes, but is not restricted to, phenomena whose markings are left in consciousness as a temporal evolution of beliefs. hypotheses, goals, subgoals, etc.

Neither traditional symbolic computationalism nor radical connectionism has much use for this distinction since all the phenomena of intelligence, micro and macro, are meant to come under their particular purview. I would like to present the case for a division of responsibility between connectionism and symbolic computationalism in accounting for the phenomena of interest. Simply put, the architectures in the connectionist mold offer some elementary functions which are rather different from those assumed in the traditional symbolic paradigm. By the same token, the body of macro phenomena seems to me to have a large symbolic and algorithmic content. A proper integration of these two modes of information processing can be a source of powerful explanations of the total range of the phenomena of intelligence.

I am assuming it as a given that much of high level thought has a symbolic content to it (see (Pylyshyn, 1984) for arguments that make this conclusion inescapable). *How much* of language and other aspects of thought require this can be matter of debate, but certainly logical reasoning should provide at least one example of such behavior. I am aware that a number of philosophical hurdles stand in the way of asserting the symbolic content of conscious thought. If one is a radical behaviorist or a non-representationalist, I can see that no advantage accrues from granting that the corpus of thought. including language and reasoning, has a symbolic structure. Saying that all that passes between people when they converse is airpressure exchanges on the eardrum has its charms. but I will forego them in this discussion.

Asserting the symbolic content of macro phenomena is not the same as asserting that the internal language and representation of the processor that generates them has to be in the same formal system as that of its external behavior. The traditional symbolic paradigm has made this assumption as a working hypothesis, which connectionism challenges. Even if this challenge is granted there is still the problem of figuring out how to get the macro behavior out of the connectionist structure.

Fodor and Pylyshyn (1987) have argued that much of thought has the properties of *productivity* and *systematicity*. Productivity refers to a potentially unbounded recursive combination of thought that is possible in human intelligence. Systematicity refers to the capability of combining thoughts in ways that require abstract representation of underlying forms. Connectionism, according to Fodor and Pylyshyn, may provide some of the architectural primitives for performing parts of what is needed to achieve these characteristics, but cannot be an adequate account in its own terms. We need computations over symbol systems, their capacity for abstract forms and algorithms, to realize these properties.

In order to account for the highly symbolic content of conscious thought and to place connectionism in a proper relation to it, Smolensky (1988) proposes that connectionism operates a lower level than the symbolic, a level he calls *subsymbolic*. He also posits the existence of a *conscious processor* and an *intuitive processor*. The connectionist proposals are meant to apply directly to the latter. The conscious processor may have algorithmic properties, according to Smolensky, but still a very large part of the information processing activities that have been traditionally attributed to algorithmic architectures really belong in the intuitive processor.

A complete connectionist account in my view needs to account for how a sub- or nonsymbolic structure integrates smoothly with a higher level process that is heavily symbolic. There is the additional problem that an integrated theory has to face. Thought could be epiphenomenal. However, we know that the phenomena of consciousness have a causal interaction with the behavior of the intuitive processor. What we consciously learn and discuss and think affects our unconscious behavior slowly but surely, and vice versa. What is conscious and willful today becomes unconscious tomorrow. All this raises a more complex constraint for connectionism: it now needs to provide some sort of continuity of representation and process so that this interaction can take place smoothly.

Connectionist and symbolic computationalist phenomena, in my view, have different but overlapping domains. The basic functions that the connectionist architecture delivers are of a very different kind than have been assumed so far in AI, and thus computationalist theories need to take this .nto account in their formulations. A number of investigators in AI who do theories at this higher level correctly feel the attraction of connectionist style theories for some parts of their theory making. I have acknowledged the power of the connectionist claims that for some information processing phenomena, there exist nonalgorithmic schemes which make fewer (and different) commitments in terms of representational content. Where the impact of connectionism is being felt is in identifying some of the component processes of overall algorithmic theories as places where a connectionist account seems to accord better with intuitions. As I said earlier, retrieval and matching operations and low level parameter learning are places where I would think the higher level theories may choose connectionist alternatives if the fine points of performance are of theoretical importance. But, even here one should be careful about putting too much faith in connectionist mechanisms per se. As I have said several times in this section, the power for even these operations is going to come from appropriate encodings that get represented connectionistically. Thus, while memory retrieval may have interesting connectionist components to it, the basic problem will still remain the principles by which episodes are indexed and stored, except that now one might be open to these encodings being represented connectionistically. For example, I am in complete sympathy with the suggestion by Rumelhart, Smolensky, McClelland and Hinton (1986) that a schema or a frame is not explicitly represented as such, but is constructed as needed from more general connectionist representations. This does not mean to me that schema theory is only a macro approximation. Schema, in the sense of being IP abstractions needed for certain macro phenomena, is a legitimate conceptual construct, for encoding which connectionist architectures offer a particularly interesting way.

With regard to general AI and connectionism's impact on it, I would like to say, as H. L. Mencken is alleged to have said in a different context, "There is something to what you say, but not much." Much of AI (except where microphenomena dominate and computationalist AI is simply too hard edged in its performance) will and should remain largely unaffected by connectionism. I have given two reasons for this. One is that most of the work is in coming up with the information processing theory of a phenomenon in the first place. The more complex the task is the more common are the representational issues between connectionism and the symbolic paradigm. The second reason is that none of the connectionist arguments or empirical results show that the symbolic, algorithmic character of thought is either a mistaken hypothesis, purely epiphenomenal or simply irrelevant.

My arguments for and against connectionist notions in this section are not really specific to connectionist architectures that have been proposed. The arguments apply to alternatives in the nonsymbolic paradigm. e.g., all sorts of analog computers. Connectionist style architectures, especially those that deny modeling the brain level, often seem to have an air of arbitrariness about them, since it is then not clear what the constraints are: why that rather than something else? But, in fairness, these architectures ought to be viewed as exploratory, and in that sense they are contributing to our understanding of the capabilities and limitations of alternatives to the symbolic paradigm.

It seems to me that we need to find a *modus vivendi* between three significant insights about mental architectures:

- (i) A large part of the relevant content theory in AI has to do with the *what* of mental representations. I have called them IP abstractions.
- (ii) Whatever one's position on the nature of representations below conscious processes, it is clear that processes at or close to that level are intimately connected to language and knowledge, and thus have a large discrete symbolic content.
- (iii) The connectionist ideas on representation suggest how nonsymbolic representations and processes may provide the medium in which thought resides.

3.9. Transition to the Rest of the Paper

Connectionism has been important to my discussion not because its technical accomplishments have thrown down a challenge to symbolic approaches to AI, but because it replays one side of a long-running debate about the nature of the relationship between mind and brain. In spite the hopes of some of its supporters in philosophy, connectionism will not banish mind-talk, which is essentially representational and symbolic.

I now want to pick up the main thread of this paper. I want to review AI theories of the last two decades and see if a view of what makes intelligence a coherent computational phenomenon can be constructed. In view of the arguments in this section, my discussion will deal with the macro phenomena of intelligence.

In much of what follows, I will be talking within the symbolic paradigm for the reasons that I have described *in this section.*

4. Current Styles of Theory-Making in AI

From the viewpoint of the paradigms of intelligence that characterize the current work in AI, at the end of the first decade the computationalist paradigm emerged as the preferred one for much theory making. I see the research in this paradigm in the next two decades until the contemporary period as belonging to one of three broad groups of stances towards what a computational account of intelligence should look like. This characterization I am about to give is a personal one, and not (yet) part of the field's own self-consciousness; that is, it is really in the form of a thesis of what has been going on in the field from the perspective of a science of intelligence, and where people have been looking for answers, and what sorts of (often unconscious) assumptions about the nature of intelligence are implicit in these theories. Another caveat is that these theories are not mutually exclusive, (i.e., some important ideas ideas appear in more than one approach, but with a flavor relevant to the approach), but constitute different ways to talk about the stuff of intelligence, and different answers to its nature. They are:

- I. *Architectural theories*
- II. *Abstract logical characterization of an agent's knowledge, and inference mechanisms that operate on this representation*
- III. *Theories that emphasize generic funtional processes in intelligence. Each of these processes generates efficient inferences for a type of information processing task, and gives importance to organizational issues as a major source of this efficiency.*

In the next several sections, I consider each class of theories and examine their assumptions about the nature of intelligence.

4.1. Architectural Theories

Architectural theories, of which the Production System theory (Newell, 1973) is the most prominent, are a result of a search for a level of a machine at which *intelligence qua intelligence emerges*. In this section I will argue that such theories fail to relate the architecture to intelligence in such a way that one can see the essential role played by the architecture in the emergence of intelligence. Their Turing universality often muddles the issue. They tend to foster a search for solutions at the architecture level, when a more problem-specific solution would be more appropriate. These architectural level solutions often have a "syntactic" feel to them in comparison with more content-driven solutions at the higher level.

In architectural theories the solution to the problem of intelligence is to provide a computational architecture which is intrinsically seen as the source of intelligence. Below that level of abstraction are presumed to lie matters of implementation out of non-intelligent processes, whereas above that level, i.e., the *content* of such architectures, are agent-specific particulars of the world, the domain, etc., not of intrinsic interest to a theoretician of intelligence, because the architecture is supposed to provide *all* the necessary elements of intelligence. In particular, it is very important for these kinds of theories that the architecture being proposed be a *unitary* architecture. By unitary, I mean one level rather than multiple levels of architecture, each of which contributing some aspect of intelligence.

Let me be concrete, so that it may be clearer what I mean to include in this classification. Typically, intelligence as a whole or a large class of problems within intelligence is treated as solvable by representing the information processing activity in some very general scheme: production rules, logical clauses, frames, semantic networks, or whatever. Then a basic class of inference or information processing activity on these representations is defined: forward or backward chaining in production systems, truth maintenance (Doyle, 1979) or resolution processes for logical sentences. various kinds of interpreters for frames or semantic networks, and so on. Then any particular problem, say diagnosis or design, is solved by attempting to program a solution using the underlying architecture. For example, diagnosis would be treated as particular example of truth maintenance, backward chaining, propagation of values in a network, or whatever the processes the underlying architectures directly support. Thus the diagnostic problem is reduced to programming in a given architecture. I will frequently use the production system architecture as an example to make our points regarding architectural theories. The comments are meant to apply to all unitary architecture theories.

In production systems, intelligence is viewed as a rule processor. What the production systems are actually implemented on -- say Lisp machines in AI or neural structures in natural intelligence -- are themselves unimportant from the viewpoint of intelligence, though they may be relevant from considerations of engineering issues such as speed. Similarly, in this perspective, what particular production systems contain, e.g, rules about liver diseases or about Vax configurations, is *per se* not a subject matter of the science of intelligence. The inference mechanisms at the level of the architecture -- the various chaining schemes and conflict resolution -- are however the subject matter of this science, since those mechanisms are intrinsically and intimately related to the architecture.

You can substitute your favorite architectural theory of intelligence-- frame systems, Logic architectures, belief networks, etc. -- for the production system example above, and come up with a similar analysis. The point is that the theories propose a *unitary* architecture as a privileged level to capture intelligence. Whenever anyone says, "The mind is a ....," you know that person is proposing an architectural solution to AI. Now, among these architectures, some of them may have more psychological evidence than others, but the problems associated with seeking the seat of intelligence at *one* level of the architecture.

It is true that each of these architectures has been used to build impressive systems that perform some task requiring intelligence. However, since these architectures are Turing-universal[7], i.e., any algorithm can be implemented as a program for this architecture, it is often hard to know if the architecture *per se* is performing important work. It may merely describe an appropriate implementation. The architecture, *a priori*, does not distinguish tractable solutions from intractable ones. It does not identify good vs bad ways of organizing knowledge. Also, because it is a unitary architecture, it necessarily omits as constructs of the architecture important, higher level information-processing distinctions that are needed to give an adequate functional description of intelligence and *which may require architectural support of their own.* A well-known proponent of rule-based architectures said during a lecture, "Common sense is just millions and millions of rules." One might well respond, "Yes, and *War and Peace* is just thousands and thousands of sentences."

The case of metarules in production systems can be used to illustrate this point. In knowledge-based systems work, the rule architecture was originally offered as as advantageous because it permitted a knowledge-base of domain facts, and an inference engine which implemented the control of reasoning using the facts in the knowledge base. It turned out that the control at the rule level was inadequate to perform a wide variety of tasks, so a supposedly domain-specific knowledge base increasingly acquired rules whose role was to provide the requisite control behavior and focus in problem solving. That is, it was seen that *there were phenomena essential to intelligent behavior that were above the rule level of architecture, but were not merely a collection of agent-specific world facts.* At this point, the idea of *metarules* was introduced, where each metarule was a domain-independent[8] control rule that helped organize the knowledge base for a particular type of problem solving activity. However, while the metarule was a way of using the rule architecture to *implement* a control strategy in rule architectures, there was no obvious reason why the underlying lesson, *viz.*, that we need to organize the control and knowledge base so that the facts in it were used in a certain way for certain kinds of tasks, was limited to the rule architecture. Rather the need for control knowledge in addition to domain knowledge is a lesson that can be applied to any unitary architecture. The control vs domain knowledge distinction is a high level statement about the content or role of knowledge and not a syntactic statement about a particular architecture. If the desired problem solving behavior did come about as the result of the metarule, the rule architecture was not, *a priori* responsible; instead, the credit should go to the particular control knowledge represented by the meta rule. Thus the commitment to the unitary rule architecture at first suppressed the higher level distinctions at the level of the control strategies implicit in the different metarules. While metarules had the germ of the idea that intersting control issues were being given short shrift in rule-based approaches, it is interesting that until Clancey's work (Clancey,

[7]There is nothing in this argument that turns on whether or not intelligence can be accounted for by Turing-computable functions only.

[8]The word "domain" is a possible source of confusion. In AI, the term has come to refer almost exclusively to what one might call specific subject matter-- or a collection of facts about a miniworld -- such as medicine, whereas in many philosophical discussions about mind. I have seen the term refer to any generic faculty such as visual perception or even tasks such as natural language understanding or diagnostic reasoning. In this paper, I will use the term as it has become familiar in AI. The assumption is that no theory of intelligence need specifically deal with medical facts per se, but it will need to deal with phenomena such as diagnostic problem solving or natural language understanding.

1984), the syntactic aspect of metarules, not their content, dominated the discussion in rule-based system discussions. Thus much of the discussion on metarules emphasized their syntactic properties. i.e., were a contribution to rule programming. This concern with converting higher level content phenomena into syntactic solutions at the architecture level is what I mean by suppressing higher level distinctions. Another example, with somewhat serious consequences in my view, is also from the rule-based architecture example. One of the problems that arises in this architecture when the knowledge base is very large is that a large number of rules "match" a given situation., with conflicting actions proposed by each of the rules. The processor has to choose one of the rules and pursue the consequences in terms of the inferences that follow. A new problem called the "conflict resolution" problem was formulated, and a family of essentially syntactic resolution strategies were proposed, such as "Choose that rule which has more left hand side terms matching, " "...has more goals on the right hand side", or "...has a higher certainty factor," etc. In truth, I claim that conflicts of this type are artifacts of the architecture, and there are higher level organizational phenomena that actually constrain only a portion of the knowledge base to be active in the first place, thus obviating the need for syntactic conflict resolution strategies.

While I have used rule architectures as examples in my discussion above, the points made are more general, and apply to unitary architectures in general. Retrieval theories based on semantic networks tend to explain differential retrieval by positing that they are due to distances measured in the number of links rather than the types of links and the knowledge embedded in the nodes and links. These unitary architectures encourage theory making at the wrong level of abstraction.

The end of this section is probably a good place to make a number of clarifications.

- The above arguments are not directed against the existence of appropriate architectures on which to implement intelligent information processing systems. In principle, it is even possible to hold that all higher level phenomena of importance can be implemented on the architecture at one particular level in such a way as to give some significant performance or construction advantages. What is being argued is that *seeking a unitary architecture as the single answer to what makes intelligence* is fraught with the problems that I have described, and misstates the nature of intelligence. I doubt that there is one level which can be identified with the emergence of intelligence. On the other hand, the discussion about whether rule architectures or alternate architectures properly capture the human information processor at one level of abstraction is not vacuous: it *is* a relevant and useful question. It is also important to state that researchers who have been investigating these architectural questions, e.g., Newell and Simon in the case of rule- architectures, do not themselves necessarily reduce all higher level issues in intelligence to syntactic aspects of their particular architectures. It is an empirical fact, however, that these architectures do foster a unidimensional view of intelligence, resulting in important higher level questions being ignored or treated as mere programming issues at the architecture level as in the example of metarules that I discussed.

- I mentioned frame architectures as another example of unitary architectures. This is not an argument against "frames" as a functionally useful construct for intelligence; in fact, in a Section 4.5, I will offer the frame theory as an example of the right kind of theory-making in AI. In all of the discussions in this section, my argument has been against a style of theory-making which converts such a theory into a universal architecture.

- The arguments in this section are not meant to be against the *idea* of architectures to support intelligent computation. In Section 4.5 I argue that intelligent information processing comes as a variety of functional types, each of which is supported by a local "mini" architecture. Form follows function, in this as in many other designs, natural or artifactual. Thus the suggestion is that instead of fixing on one form (even if true at some level), it is more productive to identify functions and then look for forms to support them.

4.2. Theories of Intelligence Based on Abstract Logical Characterization of Agents

In many circles some version of logic is thought to be the proper language of characterizing all computation, and by extension, intelligence. By logic is meant a variant of first order predicate calculus or at least a system where the notion of truth-based semantics is central and inference-making is characterized by truth-preserving transformations.[9] The way logic has actually been used in AI however combines a number of distinct roles that logic can play or can be claimed for it. We can begin by noting two broad roles for it:

1. logic for abstractly characterizing and analyzing what an agent knows, *vs*

2. logic as a representation for knowledge, and logical deduction as the basic information processing activity in intelligence.

First of all, there seems to be a general tendency, even among those who do not adopt the logic paradigm for knowledge representation and inference, to concede to logic the status as the appropriate language for the abstract characterization of an agent, i.e., for meta-AI analysis. While *this seems like one good possibility, this does not seem to me the only or even a compelling possibility.* Standing outside an intelligent agent, one can take two distinct abstract stances toward it: the agent as a performer of functions, or the agent as a knower of propositions. There are AI proposals and work that correspond to both these viewpoints. In Marr's work on vision (see Section 3.6), e.g., the agent is characterized *functionally*, i.e., by an information processing task that transforms information of one kind into that of another kind. On the other hand, the tradition in logic-based AI is one of attributing a body of knowledge to the agent. It is certainly not obvious why the knowledge view should necessarily dominate even at the level of abstract characterization. I will hold later in this paper that the functional view has superior capabilities for abstract specification of an intelligent agent.

The idea of abstract characterization of an intelligent agent through logic was first detailed by McCarthy and Hayes (1969) where they proposed the now-famous *epistemic-heuristic* decomposition of an actual intelligent agent. This distinction has echoes of the Chomskian competence/performance distinction in language. (See (Gomez and Chandrasekaran, 1981).) The agent as a knower is characterized by the epistemic component. What kinds of knowledge are to go into the epistemic component is not clear, but one would think that would depend on the theorist's view of what kinds of knowledge *characterize* intelligence. Thus it must represent a theory of the ontology of the mental stuff. The heuristic part is that part of the agent which actually makes him an efficient information processor, using the knowledge in the epistemic part to solve problems and do whatever intelligent agents do. An analogy would be that a calculator's epistemic part would be the axioms of number theory, while its heuristic part would be the particular representations and algorithms. This example also makes clear the relationship of the epistemic/heuristic distinction to the competence/performance distinction of Chomsky. McCarthy and Hayes proposed that the epistemic part be represented by a logical calculus, and in fact discussed the kind of logic and the kinds of predicates that would be needed for adequacy of representation as they conceived the epistemic part. In this attempt to separate the *what* of intelligence with the *how* of implementation, the McCarthy-

[9]I am sure this characterization is not accurate in details. I am told that there things such as "imperative logics" where some of the above characterization might not hold good. Within AI, nonmonotonic logic relaxes the truth-preserving requirement in transformations. I believe that the thrust of my arguments nevertheless remain valid.

Hayes proposal follows a more general idea in computer science, but identifies the *what* with the propositional knowledge of the agent. This is not, however, neutral with respect to consequences.

Now it is interesting to note that the epistemic/heuristic distinction as a means of separating the essential content of an agent from the implementation details and "mere" efficiency considerations is independent of logic as a representation. As just mentioned, this kind of separation is a strong tradition in computer science, at least in proposals about software engineering, if not in practice. All that the epistemic/heuristic distinction demands is that the essential characterization of an information processor be kept separate from the implementation details. Logical representation of the epistemic part is only one alternative for doing this. As I mentioned, within AI, an alternative to McCarthy and Hayes' suggestion is Marr's proposal that this distinction be carried out by separating the information processing task (the input–output specification), the algorithm carrying out the task, and the mechanism by which the algorithm is implemented.

Even more important is that there is no self-evident way of deciding, in a theory-neutral fashion, exactly what is epistemic, and what is merely heuristic. One person might dismiss some aspect of an intelligent agent as merely heuristic (e.g., search control knowledge which helps in efficiency), while another person's theory might hold that that is precisely where the secret of intelligence lies, and thus would make it part of the epistemic component. The epistemic/heuristic distinction does not force one into an agreement about what entities ought to end up in which component. In fact, in some logical theories in AI, some important control phenomena have been moving from the heuristic component to the epistemic component, as a consensus builds in AI that a certain phenomenon is *really* not simply heuristic, but part of the stuff of intelligence. A good example of this is the development within the logic camp of families of *default or nonmonotonic logics*. It is interesting to trace the history of default logics within the logic paradigm in AI.

Now, Minsky's paper on frames (Minsky, 1975) argued against the "atomic" stand about knowledge that logic-based AI theories took, and claimed that chunking of knowledge into "frames" had a number of useful properties. In particular, if frames could stand for stereotypes of concepts, it was possible to do a form of default reasoning, where in the absence of specific information to the contrary, a reasoning system would assume default values associated with the features of a frame, thus allowing missing information to be plausibly inferred, greatly decreasing storage requirements (only non-default values, i.e., exceptions, need to be stored), and increasing retrieval speed. At first blush, all these useful properties were "heuristic" aspects, i.e., how to get the computation done faster, and thus one would think not of intrinsic epistemic interest. However, when the consensus started to develop that this form of default reasoning was one of the essential aspects of being intelligent, defaults became part of the epistemic component of AI. That is, theorists started hypothesizing the existence of something called "default reasoning" or "nonmonotonic logic" in order to account for this phenomenon in a rigorous way. Similarly, as builders of AI systems discover the utility of organisational constructs such as "plans" (which are abstract, partial solutions to problems, stored by the agent, typically indexed by the goals they help achieve), one finds that new epistemic theories such as *plan logics* get proposed and investigated. It is almost as if the lowly "heuristic" component is in fact what the action in AI is often about, while the epistemic part appropriates the nuggets of organisational wisdom that research in the heuristic component identifies.

Now of course there is nothing wrong with this as a way of making scientific progress: Let the system builders discover phenomena experimentally and let the theoreticians follow up with formalisation. However, the eventual success of this kind of formalization is questionable. Is there in fact a set of inference rules that compactly characterize all and only default inferences? I propose that terms such as "nonmonotonic logic" are reifications: a complete account would require specification of a much larger set of rules than what is normally thought of as inference rules; so large as to be virtually coextensive with the entire set of distinct types of functions in which frames get used: scripts, plans, etc.

The distinct notions of abstractly characterizing an intelligent agent, viz., the epistemic heuristic distinction, and logic as a *representation* for the epistemic component, are often conflated. I just commented on how uncertain the actual allocation of information to the epistemic and heuristic components can be. Now I need to make some remarks on logic as a representation for the epistemic component.

4.3. Logic for Representation

The proposal to use logic for representation of knowledge could be in the service of two rather different purposes: one, in order to reason *about* the agent and two, to model the reasoning *of* the agent. The former is in the spirit of certain ideas in computer science where a program may be in any appropriate language, but reasoning about the program, e.g., to establish its correctness. is often done using logic. However, in practice, almost all use of logic as knowledge representation in AI has been in the service of the latter, i.e., to actually create reasoning agents.

Logic as knowledge representation makes a serious commitment to knowledge as propositions, and to True/False judgements as the basic *use* of knowledge. It is also closely connected to the belief that the aim of intelligence is to draw *correct* conclusions. In this view, what human beings often do, e.g., draw plausible, useful, but strictly speaking logically incorrect conclusions, is interesting as psychology, but that only shows up humans as approximations to the ideal intelligent agent, whose aim is to be *correct*.

A little digression about the nature of the ideal intelligence may be appropriate here. I believe that there has been a problem in AI due to two different senses of the word "intelligence." There is the technical sense of intelligence as the information processing activities engaged in by, possibly among others, human beings, and which is the goal of AI to understand and capture. There is another sense in which intelligent really refers to "very intelligent," i.e., some one who has been especially impressive in his or her cerebration. At least in the current western milieu, this latter quality would be denied to someone who, after all the work of thinking, was not correct in the conclusions that were drawn. Ever since late 19th century, when the foundations of mathematics showed cracks and there were considerable worries about how to be sure if conclusions were correct, which in turn pushed symbolic logic to its current rich technical accomplishments, logical reasoning has been equated with the real *test* of thought, *vide* the title of Boole's book, *Laws of Thought.* In addition, the content of consciousness seems to include a series of propositions, some of them beliefs, and at least for a certain kind of theorist, it seems entirely natural to model thought itself as basically manipulation of propositions and generation of new ones. In this view of thought, stream of consciousness imaginings, half-formed ideas, vague sensations at the back of the mind, how a certain idea suddenly came to occupy consciousness from the depths of the mind, etc., etc., do not count as serious subject of study from the viewpoint of intelligence as such. Hence the almost unconscious equation of thought with logical thought, and the natural attempt to seek in logic the language of representation and construction of the idealized agent.

Now, is "truth" in fact the right kind of basic interpretive framework for knowledge? Or are notions of functional adequacy, i.e., knowledge that helps to get certain kinds of tasks done, or the related notions of plausibility, relevance, etc. more effective in capturing the way agents in fact use knowledge? My 16-month old daughter, when shown a pear, said, "apple!." Is it more than mere parental pride that makes me attribute a certain measure of intelligence to that remark, when, viewed strictly as an utterer of propositions, she told an untruth? What kind of a theory of intelligence can explain that her conclusion was adequate for the occasion: she could get away with that error for most purposes -- she could eat the pear and get nourishment, e.g. -- while an equally false proposition, "It's a chair," would not give her similar advantages?

A number of theoretical advantages have been claimed for logic in AI, including *precision* and the *existence of a semantics*. The problem is that the semantics are not at the most appropriate level for the problem at hand, and logic is neither a unique nor a privileged way to be precise.

4.4. *Intelligence Has Other Functions Than Correctness*

Laws of justification are not identical to laws of thought, Boole notwithstanding. While it would be useful for an intelligent agent to have the former laws and apply them appropriately, those laws alone cannot account for the power of intelligence as a process. It seems highly plausible to me that much of the power of intelligence arises not in its ability to lead to correct conclusions, but in its ability to direct explorations, retrieve plausible ideas, and focus the more computationally expensive justification processes where they are absolutely required. Thus the power of intelligence really resides in what has been called the heuristic part, and theories of intelligence will need to be theories of that part of the decomposition, the part that is most concerned with computational feasibility. This is why organizational theories such as the frame theory, planning, and theories of memory, find their important ideas migrating to the epistemic side, which by definition, in the logic framework, is supposed to worry about the real essence of intelligence. What is interesting is that the pressure of discoveries in the heuristic side comes from efforts to actually *construct* intelligent artifacts. To the extent that explanation of intelligence as a computational phenomenon is treated within AI as the capability to construct intelligent artifacts, it is significant that it is this so-called heuristic side that has been the source of important discoveries about how the intelligent information processing can be controlled. Abstraction in the manner proposed by those who advocate logic *separates knowledge from its function*, and this leads to missing important aspects of the form and content of knowledge.

It is often argued that the epistemic/heuristic distinction is tactical: get the terms needed right, before worrying about how to actually use them in reasoning. For example, before building, say, common sense reasoners, let us get all ontology of common sense reasoning right: "know," "cause," etc. The thrust of my argument is that, as a rule, a use-independent study of such terms is likely to make distinctions that are not needed by the processing part and miss some that are.

This is not to say that logic, as a set of ideas about *justification*, is not important to intelligence. How intelligent agents discover justifications and how they integrate them with discovery procedures for a final answer that is plausibly correct, and how this done in such a way that the total computational process is controlled in complexity are indeed questions for AI as an information processing theory of intelligence. In this view logic is a brick intelligence builds, rather than the brick out of which intelligence is built.

The laudable goal of separating the knowledge necessary for intelligence from the implementation details needs in my opinion to be achieved by concentrating on the functional characteristics of intelligence. This brings me to the third set of theories.

4.5. *Generic Functional Theories of Intelligence*

The theories that I discuss in this section identify a generic, functional property of intelligence which is used to solve a "natural kind" of cognitive problem. Examples of such theories are: the GPS means-ends theory of problem solving (Newell and Simon, 1972), the Frame theory of knowledge organisation (Minsky, 1975), Schank's CD, Script (Schank and Abelson, 1977) and memory theories (Schank, 1982), our own work at Ohio State on generic types of problem solving (Chandrasekaran, 1983; 1986; 1987). These theories typically emphasize some *organizational* aspect, which facilitates some particular *class of inferences*, or *computations*, or *constructions* in a computationally efficient way. An abstract description of these processes would be replete with terms that carry an information processing strategy connotation, such as *default*, *goals*, *subgoals*, *expectations*, *plans*, and *classification*. Knowledge of an agent is encoded using such terms. Each of these processes -- the inference mechanisms along with the knowledge structures -- constitute what we call a generic information processing strategy. Each captures a functional unit of intelligence as a process, and is generic in the sense that it is domain-independent.[10]

In this section, I briefly discuss some of the well-known theories of this genre. In particular, I consider such strategies in knowledge-based problem solving.

Let me recapitulate some of the theories that I mentioned at the beginning of this section.

- The General Problem Solver. GPS says that a generic process available to intelligence is to treat problems in a goal/subgoal manner. In this way, the ends, i.e., the goals, of the problem solver, are matched to the means, the operators available. The agent has to have knowledge organized in a certain way, which gives information about relationship between goals, operators, etc., and a particular inference process (or control regime) called the *means-ends algorithm* is needed to use this knowledge to solve the problem. Note that the means-end method is not implicit in the problem statement, i.e., a purely logical analysis merely would specify a problem space in which the solution would lie. On the other hand, means-ends is not a particular solution algorithm for a particular problem. It is a generic information processing strategy that can be used provided that knowledge is available in a certain form.

- The frame theory. It proposes that the generic functional properties of "chunking" and "stereotyping" are important phenomena in knowledge organization. These features of the organization enable an important type of inference process called *default reasoning* to take place by using the default expectations implicit in the idea of stereotypes. Chunking and stereotyping are computationally efficient mechanisms, both for memory and for processing.

- Schank's *conceptual dependencies* and *script* theories. These are in the spirit of frame theory, but they propose contentful additions that are appropriate for certain classes of problems. CD's are particular kinds of chunks that represent action stereotypes, and enable default inferences about actions associated with verbs to be made efficiently during natural language understanding. Scripts are frames representing stereotypical action sequences, which enable expectations to mediate understanding. Both these notions (and other similar constructs in Schank's theories of memory organization) use expectations arising from stereotypes of phenomena as efficient means of computing some information about the situation.

- Plans: these are compiled, abstract, partial solutions to problems, indexed by goals. Agents which use them cut down the search for solutions enormously, again a "heuristic" advantage, but mirrored in knowledge so deeply that it becomes epistemic in character. See (Miller, et al, 1960).

- Memory Organization & Retrieval: Agents index, store and retrieve relevant events of their experience in such a way that they provide, among other capabilities, a method of reasoning (by analogy with past even[illegible] g.) The work of Schank's group proposes a number of types of memory chunks an[illegible]exing schemes.

[10]Please see the earlier footnote on the word "domain" here.

- Generic Tasks: In our own work on knowledge-based reasoning, we have proposed a number of information processing strategies, each of which is characterized by knowledge represented using strategy-specific primitives, and organized in a specific manner. Each of the strategies also employs a characteristic inference procedure which is appropriate to the task. By showing how these strategies help in solving a computationally complex problem such as diagnosis, I hope to suggest how strategies of this kind characterize intelligence. The role of a specific set of generic strategies in diagnosis is the subject of the Section 4.6.

4.6. Generic Information Processing Strategies in Diagnostic Reasoning

Formally, the diagnostic problem can be defined as follows: Find a mapping from the set of all subsets of observations of a system, to the set of all subsets of possible malfunctions, such that the malfunctions in the subset best explain the observations.

A mathematician's interest in the problem would be satisfied once it can be shown that under certain assumptions this task is *computable*, i.e., an algorithm exists to perform this mapping. He might even further wish to derive the computational complexity of this task for various assumptions about the domain and the range of this mapping. This directly leads to AI algorithms of a set-covering variety for diagnosis (Reggia, et al, 1985).

A logician would consider the solution epistemically complete if he can provide a formalism to list the relevant medical facts and formulate the decision problem as deducing a correct conclusion. Some diagnostic formalisms, such as the ones based on truth maintenance systems, view the diagnostic problem as one more version of truth-maintenance activity (Reiter, 1987).

Now, each of these methods is computationally quite complex, and without extensive addition of knowledge as "heuristics", the problem cannot be solved in anything resembling real time. It is clear, however, that the abstract problem is one that faces intelligent agents on a regular basis: *how to map from states of the world to their explanations?* From the tribesman on a hunt who needs to construct an explanation of observations on the jungle ground to a scientist constructing theories, this basic problem recurs in many forms. Of course, not all versions of the problem are in fact solved by humans, but many versions of the problem, such as medical diagnosis, are solved quite routinely. Presumably something about the agent as an intelligent information processor directly plays a role in this solution process.

Because of our concern in this paper with the structure of intelligence, instead of looking for solutions to this problem in particular domains (such as simple devices, where perhaps tractable algorithms -- e.g., direct-mapping tables that go from symptoms to diagnoses -- might exist and be programmed), let us ask the following question: *What is an intelligence that it can perform this task?* That is, we are interested in the relation between mental structures and the performance of the diagnostic task. The distinction that we are seeking can be made clearer by considering multiplication. Multiplication viewed as a computational task has been sufficiently studied that very fast and efficient algorithms are available, and are routinely used by today's computers. On the other hand if we were to ask, "How does a person (e.g., an arithmetic prodigy) actually perform multiplication in the head?", the answer will be different from the multiplication algorithms just mentioned. The answer would need to be given in terms of how the particular problem is solved by using more generic mental structures. Now, of course, the answer would differ depending upon one's theory of what those mental structures are.

I have already indicated what kinds of answers to this question would be fostered by unitary architectures: In rule-based architectures, the problem solver will simply need to have sufficient number of rules about malfunctions and observations, frame–theorists would propose that diagnostic knowledge is represented as frames representing domain concepts such as malfunctions, etc. The inference methods that are applicable to each of the above are fixed at the level of the functional architecture: some form of forward or backward chaining for rule systems, and some form of inheritance mechanisms and embedded procedures for frame systems. I have argued elsewhere how this level of abstraction for control is at too low a level to perspicuously encode the inference processes that apply at the level of the task, i.e., diagnosis. (Since all these architectures are computation– universal, they can be made to encode whatever diagnostic algorithm the designer has in mind, but the ideas underlying the algorithms are "lost" as code at these levels, rather than explicitly supported by the architectural constructs.) This level of representation suppresses what is distinctive about diagnosis as a set of domain–independent information processing strategies. See our earlier arguments regarding syntactic solutions at the architectural level.

In our work on knowledge–based reasoning, we have identified several generic strategies, each with a well–defined information processing function. Each of them uses knowledge in certain forms, organized in certain ways, and employs inference strategies that are appropriate to the task. We have described, in a series of papers, how these strategies can be used in different combinations to put together diagnostic or design systems. In the rest of this section, I want to describe briefly how three strategies of the above–described type can come together to solve a number of real world versions of the diagnostic task.

In many domains knowledge is available in the form of malfunction hierarchies (e.g., disease hierarchies in medicine) and for each malfunction hypothesis in the hierarchy, a mapping from observations to the degree of plausibility of hypothesis can be done using a strategy of *concept matching*. In concept matching, a concept is matched to data by a hierarchy of abstractions, each of which produces a degree of local match. In such domains, the diagnostic problem can be *decomposed into three subproblems* (Chandrasekaran, 1986; Josephson, et al, 1987):

1. Hierarchical classification: A classification process on the diagnostic hierarchy is invoked. At the end of the classification process a set of tip nodes of the diagnostic hierarchy are "established" to some degree of plausibility, each explaining a set of observations. In the medical domain, these tip nodes will correspond to specific diseases or in the case of mechanical systems, they may be malfunctions of specific components.

2. Concept–matching: Each of the hypotheses in the classification process is evaluated by appealing to the appropriate concept–matching structures which map from relevant data to symbolic confidence value for that hypothesis.

3. Abductive Assembly: The classification process (in conjunction with the concept matchers) terminates in a small number of highly plausible hypotheses, each explaining a subset of observations. An *abductive assembly* strategy, which uses knowledge about intercation among malfunctions, can be used to assemble a subset of them into the a composite hypothesis that best explains all the data.

Under the right conditions of knowledge availability, each of the above strategies is computationally tractable. In hierarchical classification, entire subtrees can be pruned if a node is rejected. The mapping from data to concept matching can be done by hierarchical abstractions giving concept matching a similar computational advantage. Abductive assembly can be computationally expensive, but if some other process can prune the space and generate only a small number of hypotheses to begin with, then its computational demand can be kept under control. This is precisely what hierarchical classification does in the above scheme.

The original intractable problem has been converted, by a series of information processing strategies and by appropriate types of knowledge and control, into a tractable one, *for those versions where knowledge of the required form is available.*

Classification as a strategy is ubiquitous in human reasoning because of the computational advantages of indexing action knowledge over equivalence classes of states, instead of the states themselves. How classification hierarchies are created -- from examples, from other types of knowledge structures, etc. -- requires an additional set of answers. I have discussed elsewhere (Sembugamoorthy and Chandrasekaran, 1986) how knowledge of the relationships between structure and the functions of components, i.e., how the systems work, can often be used to derive such malfunction hierarchies. These processes in turn are generic, requiring knowledge in specific forms and using appropriate but characteristic inference strategies.

Let me make something quite clear at this point. The claim is not that diagnosis is *logically* a classification problem, or even that all human diagnostic reasoning uses classification as one of the strategies. What I have attempted to show is that many version of the diagnostic problem can be. and often are, solved by having knowledge in forms that this and other generic strategies can use. If that knowledge is not available, either other strategies that can generate knowledge in that form are invoked, or other strategies that can help solve the diagnostic problem without classification hierarchies are attempted. In particular, strategies such as *reasoning by analogy* to an earlier case, or merely retrieval of similar cases and explaining the differences by adding, deleting or modifying diagnostic hypotheses are tried. In fact, as mentioned earlier, the whole collection of retrieval strategies (Schank, 1982) are themselves information processing strategies of the functional kind that I have been talking about.

4.7. Functional Theories: Heuristic Becomes Epistemic

What I have so far called functional theories within AI -- GPS, frames as stereotypes, conceptual dependency theory, scripts, and generic tasks in problem solving -- all have this in common: They all typically emphasize some organizational aspect and facilitate some particular kind of inference or construction in a computationally efficient way. In other words, computational feasibility -- the so-called heuristic component -- is built into this kind of theory making. Organization serves directly in securing computational feasibility. A direct epistemic analysis of the underlying problem would typically miss these constructs. Once you discover them you can go back and start doing epistemic analysis on them, but basically the way of discovering them is not by simply taking an epistemic stance toward them (here I use "epistemic" in the McCarthy-Hayes sense of the term).

Another important thing about this with respect to knowledge is that each of these approaches provides primitive terms for encoding the relevant knowledge. GPS proposes that some of the knowledge ought to be encoded in the form of goals and subgoals. The conceptual dependency primitives are provided from the CD theory. Our work on generic tasks has resulted in a family of languages, each of which provides primitives to capture the knowledge needed for that one of the generic strategies. In my view, these primitives constitute part of the vocabulary of the language of thought.

The search for such strategies as the basic task of research in AI in fact defines a new view of epistemics: It is the abstract characterization of such strategies and the corresponding types of knowledge. Such an abstract description of these processes would be replete with terms that carry an information processing strategy connotation, such as *default*, *goals*, *subgoals*, *expectations*, *plans*, and *classification*.

5. A proposal on the Nature of Intelligent Information Processing

I have given an overview of the three kinds of theories that have been advanced about the nature of intelligence[11]:

- architectural theories
- logical abstraction theories, and
- functional theories

and indicated a clear preference for functional theories. I would now like to generalize this preference into a proposal about the nature of intelligence.

The Proposal

Intelligence is a coherent repertoire of generic information processing strategies, each of which solves a type of problem in a computationally efficient way, using knowledge of certain types, organized in a specific way, and using specific and locally appropriate control strategies.

What is common, as intelligent agents, between Einstein, the man–on–the street, the tribesman on a hunt, and, probably, intelligent Alpha–Centaurians (if such things exist) is that they all face very similar computational problems, and the *kinds* of solutions that they adopt for these problem have an essential similarity. They all use plans, indexed by goals, as efficient means of synthesizing actions, they all use some version of scripts and conceptual dependency primitives to organize their inferences, they all use classification strategies to match actions to world states, etc., etc. Of course the strategies that we may discover by studying *human information processing* may not be -- and in all likelihood is not -- coextensive with the general class of such strategies. That would be too anthropomorphic a view of intelligence.

The task of AI as the science of intelligence is to identify these strategies concretely, and understand how they integrate into coherent wholes.

In a sense this approach can be called *abstract psychology* because it doesn't discuss a particular human being or even class of human beings. What it says is that the description of cognitive strategies provides a language in which to describe intelligent agents. And also, I think, it is consistent with the view of intelligence as a biological, evolvable collection of coherent 'kluges' that work together. So intelligence is not really defined as one thing -- architecture or function -- it is really a collection of strategies. The fact that the strategies all contribute to computational feasibility distinguishes them as a characterizable class of information processes.

Some qualifying remarks about the scope of my discussion are perhaps necessary. Almost all my discussion has emphasized cognitive phenomena in contradistinction to perceptual phenomena. Obviously the role of knowledge and control in perception is a much different issue than in cognition. In general, I have not included in my discussion what Fodor (1983) calls *input modules* (as opposed to central processes): his modules include some aspects of parsing in language, e.g. The spirit of what I say in this paper can be extended to these other phenomena, I believe. But that is a task for another day.

[11] Specifically, "about the nature of "macro" phenomena in intelligence," since I have acknowledged that connectionist-like theories may have something useful to say about the microstructure.

6. Concluding Remarks

In this final section, I would like to make some remarks about the relationship of functional theories to architectural and abstract characterization theories.

Some of the intuitions behind the architectural theories and abstract characterization theories are in fact valid and useful, but theory-making of these kinds can be enhanced by asking questions of the functional kind first. In particular, considering architectural theories first, it is probably true that there *does* exist an architecture at which mental phenomena can be particularly appropriately implemented. Certainly in the case of human intelligence there *is* some level to which the information processing architecture question can be reduced: if needed, to the neuronal information processing level: possibly to what connectionists call the subsymbolic level; preferably to the level of something like a rule architecture. Certainly I don't intend to argue against the existence of that level of the architecture and its properties. However, the content phenomena of intelligence as a computation are not expressed at that level, and require an analysis at the functional level as I have indicated.

There is another aspect to the architectural issue. To the extent that each of the strategies uses knowledge primitives, and comes with its own inference methods, a local architecture can be associated with it. For example, we have developed a family of high level architectures (or, it comes to the same thing, languages) for the generic information processing strategies that we have identified: a language called CSRL (Bylander and Mittal, 1986) supports hierarchical classification and structured concept matching, PEIRCE (Punch, et al, 1986) supports abductive assembly, and so on. The information processors built using these architectures can exchange the results of their computation with each other.

The functional orientation that I have advocated makes possible a new approach to mental architectures which are nonunitary, i.e., intelligence is conceptualized as a community of "specialists", each of which is an instance of a functional type of knowledge/inference system, communicating with other such entities. In fact our own group's work on problem solving has been implemented using precisely this notion of a nonunitary architecture. It needs to be reiterated that this notion does not argue against the whole set of them being implemented on a lower level unitary architecture, such as a rule architecture. It merely talks about the importance at the conceptual level of not being driven by the unitary architecture notion for all the reasons that we elaborated in Section 4.1 on this subject.

Similarly, the functional theories suggest a new approach to epistemics. They do not argue against the importance of characterizing intelligence independent of incidental implementation considerations (neurons vs transistors, e.g.), or of agent-specific heuristic knowledge (such as the knowledge that a particular agent might have, e.g., "When considering malfunctions in an electronic circuit, always check the power source first.") It is just that this approach proposes an alternative basis on which to make the abstract characterization. I propose that we ask, "What kinds of processes do intelligent agents perform?", rather than, "What kinds of things do they know?" as the starting point. The claim is that what they *need* to know in order to do the tasks in fact provides a new way of doing the epistemic analysis of an agent in the abstract. At the very least, functional theories provide the content information about intelligence as computation that needs to be specified abstractly.

Intelligence as we know it is (so far) a biological phenomenon, rather than a logical or mathematical phenomenon. A comparison is often made between intelligence and flight, and people who would build flying machines by basing them on birds usually come off looking not so good in this comparison. The problem with that analogy is that flying is one (rather well-defined) function.

while intelligence is not characterized by one function. A better analogy would be with understanding life: all we know about life is that it is what biological phenomena pertain to. Any particular aspect of life, e.g, self-reproduction, can be studied mathematically, but there has been precious little so far to show for such studies from the viewpoint of understanding biology. Intelligence is not only analogical to biology, but is, as a phenomenon in nature, so far exhibited in biological organism of certain complexity. The actual content of information processing phenomena of intelligence is bound to be rather complex. What is biological about the proposal in this paper is that intelligence is explained as part evolutionary, part cultural, part life-time interaction and integration of a number of elementary strategies into more and more complex strategies, but all of them are united by this basic concern with computational feasibility for generation of plausible solutions, rather than with deductive correctness. To see biological intelligence as a mere approximate attempt to achieve logical correctness is to miss the point of intelligence completely. Of course there are processes in intelligence that over a long period of time and collectively over humankind help to produce increasingly higher fidelity internal representations, but that is just one part of being intelligent, and in any case such processes are themselves subject to computational feasibility constraints. Once highly plausible candidates for hypotheses about the world are put together by such processes (such as abductive assembly that might be used in producing an explanation in science), then explicit justification processes may be used to verify them.

Acknowledgements: Support by Defense Research Projects Agency through RADC Contract F30602-85-C-0010 and by Air Force Office of Scientific Research, grant 87-0090 during the preparation of this paper is gratefully acknowledged. I would like to thank Dean Allemang, Larry Birnbaum, Tom Bylander, Ashok Goel, Vinod Goel, John and Susan Josephson, Allen Newell, Derek Partridge, Bill Punch, N. Sridharan, and Jon Sticklen for commenting on early drafts. I *know* that few of them agree with all the things that I say here, so the usual qualification that the author takes responsibility for what is said is particularly relevant.

References

Bruner, J. S. (1957). On perceptual readiness. *Psychological Review, 64*, 123-152.

Bylander, T. C., and Mittal, S. (1986). CSRL: a language for classificatory problem solving and uncertainty handling. *AI Magazine*, 7, 3, 66-77.

Chandrasekaran, B. (1983). Towards a taxonomy of problem-solving types. *AI Magazine, 4*, 1, 9-17.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert*, Fall, 23-30.

Chandrasekaran, B. (1986a). From numbers to symbols to knowledge structures: pattern recognition and artificial intelligence perspectives on the classification task, *Pattern Recognition in Practice-II*, E.S. Gelsema and L.N. Kanal (eds), Amsterdam: North-Holland Publishing Company, 547-559.

Chandrasekaran, B. (1987). Towards a functional architecture for intelligence based on generic information processing tasks. To appear in *Proceedings of the International Joint Conference on Artificial Intelligence*, Milan, Italy.

Clancey, W. J. and Letsinger, R. (1984). NEOMYCIN: reconfiguring a rule-based expert system for application to teaching. In *Readings in Medical Artificial Intelligence*. W.J. Clancey, E.H. Shortliffe (eds). Reading, MA: Addison-Wesley. 361-381.

Dennett, D. (1986). The logical geography of computational approaches: a view from the east pole. In Brand, M. & Harnish, R. M. (eds) *The Representation of Knowledge and Belief*, Tucson, AZ, The University of Arizona Press.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence.* 12:231–272.

Dreyfus, H. L. (1979). *What Computers Can't Do,* 2nd Edition. New York: Harper & Row.

Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis.* New York: Wiley–Interscience.

Fodor, J. A. (1983). *The Modularity of Mind.* Cambridge: The MIT Press, A Bradford Book.

Fodor, J. A. & Pylyshyn, Z.W. (1987). Connectionism and cognitive architecture: a critical analysis. (draft, to appear).

Gold, E. (1967). Language identification in the limit. *Information and Control* 16:447–474.

Gomez F. and Chandrasekaran, B. (1981). Knowledge organization and distribution for medical diagnosis. *IEEE Transactions on Systems, Man and Cybernetics.* SMC–11(1):34–42.

Hebb, D. O. (1949). *The Organization of Behavior.* New York: Wiley.

Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C. (1987). A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics.* To appear in the Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning.

McCarthy, J., and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4,* B. Meltzer and D. Michie (eds). Edinburgh: Edinburgh Univ. Press.

McClelland, J. L., Rumelhart, D. E., and Hinton, G. E. (1986). The appeal of parallel distributed processing. In Rumelhart, McClelland and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 1,* Cambridge: M.I.T. Press, A Bradford Book.

McCulloch, W., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics.* 5: 115–137.

Miller, G. A., Galanter, E. and Pribram, K. H. (1960). *Plans and the Structure of Behavior.* New York: Holt, Rinehart & Winston.

Minsky, M. L. (1975). A framework for representing knowledge. In *The Psychology of Computer Vision,* ed. P. H. Winston. New York: McGraw–Hill.

Minsky, M. L. and Papert, S. (1969) *Perceptrons,* Cambridge, MA: MIT Press.

Newell, A. (1973). Production systems: models of control structures. In *Visual Information Processing,* ed. W. Chase. New York: Academic Press.

Newell, A. (1980). Physical symbol systems. *Cognitive Science, 4,* 135–183.

Newell, A., and Simon, H. A. (1972). *Human Problem Solving.* Englewood Cliffs, N.J.: Prentice–Hall.

Punch, W., Tanner, M., and Josephson, J. (1986). Design considerations for Peirce, a high level language for hypothesis assembly. In *Proc. Expert Systems In Government Symposium,* Silver Spring, MD: IEEE Computer Society Press, 279–281.

Pylyshyn, Zenon W. (1984). *Computation and Cognition: Towards a Foundation for Cognitive Science,* The MIT Press, Cambridge, MA, 1984.

Reggia, J., Nau, D., Wang, P., and Peng, Y. (1985). A formal model of diagnostic inference. *Information Sciences*, 37, 227–285.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, 32,, 57–95.

Rosenblatt, F. (1962). *Principles of Neurodynamics*, Cornell Aeronautical Laboratory Report 1196–G–8. Washington, D.C.: Spartan.

Rumelhart, D.E. & McClelland, J.L. (1986). PDP Models and general issues in cognitive science. In Rumelhart, McClelland and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 1*, Cambridge: M.I.T. Press, A Bradford Book.

Rumelhart, D.E. & McClelland, J.L. (1986b). On learning the past tenses of English verbs. In McClelland, Rumelhart and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 2*, Cambridge, M.I.T. Press. A Bradford Book.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., and Hinton. G. E. (1986). Schemata and sequential thought processes in PDP models. In McClelland, Rumelhart and the PDP Research Group (eds.) *Parallel Distributed Processing, Volume 2*, Cambridge, M.I.T. Press, A Bradford Book.

Schank, Roger C. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. New York: Cambridge University Press.

Schank, R. C., and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Hillsdale, N.J.: Erlbaum.

Sembugamorthy, V. and Chandrasekaran, B. (1986). Functional representation of devices and compilation of diagnostic problem solving systems. In *Experience, Memory and Reasoning*. J. Kolodner and C. Reisbeck (eds). Lawrence Erlbaum Associates, 47–73.

Smolensky, P. (1988). On the proper treatment of connectionism. *The Behavioral and Brain Sciences, 11*, (in press).

Solomonoff, R. J. (1957). An inductive inference machine, *IRE National Convention Record*, pt. 2, 56–62.

Wiener, Norbert. (1948). *Cybernetics, or Control and Communication in the Animal and the Machine*. New York: Wiley.

Generic Tasks As Building Blocks for Knowledge-Based Systems: The Diagnosis and Routine Design Examples[1]

B. Chandrasekaran

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

June 28, 1988

[1]To appear in *Knowledge Engineering Review*.

Table of Contents

List of Figures

Abstract

The level of abstraction of much of the work in knowledge-based systems (the rule, frame, logic level) is too low to provide a rich enough vocabulary for knowledge and control. I provide an overview of a framework called the Generic Task approach that proposes that knowledge systems should be built out of building blocks, each of which is appropriate for a basic type of problem solving. Each generic task uses forms of knowledge and control strategies that are characteristic to it, and are in general conceptually closer to domain knowledge. This facilitates knowledge acquisition and can produce a more perspicuous explanation of problem solving. The relationship of the constructs at the generic task level to the rule-frame level is analogous to that between high level programming languages and assembly languages in computer science. I describe a set of generic tasks that have been found particularly useful in the constructing diagnostic, design and planning systems. In particular, I describe two tools, CSRL and DSPL, that are useful for building classification-based diagnostic systems and skeletal planning systems respectively. I describe a high level toolbox that is under construction called the Generic Task toolbox.

1 Need for Task-Specific Tools

The current generation of knowledge-based system (KBS) languages -- those that are based on rules, frames, or logic - do not distinguish between different types of knowledge-based reasoning. For example, one would expect that the task of designing a car would require significantly different reasoning strategies than the task of diagnosing a malfunction in a car. However, these methodologies apply the same strategy (fire the rules whose conditions match, run resolution engine on all propositions etc.) to both design and diagnosis, as well as any other task. Because of this, it has been argued that these methodologies, although useful, are rather low level with respect to modeling the needed task-level behavior. In essence, these systems resemble an assembly language for writing KBS's. While they are obviously useful, clearly approaches that more directly address the higher level issues of knowledge-based reasoning are needed for the next generation of AI system development.

One example of a higher level approach is the *generic task* (GT) [9, 10, 12]. The aim here is to identify "building blocks" of reasoning strategies such that each of the types is both generic and widely useful as components of complex reasoning tasks. We have identified a number of such generic strategies, which together capture the functionality of a large portion of current expert systems. Each generic task[2] is characterized by:

1. The kinds of information it takes as input for the task and the information produced as a result of performing the task. This defines the functionality of the task.

2. A way to represent and organize the knowledge that is needed to perform the generic task. This includes a vocabulary of knowledge types, i.e., knowledge constructs that the representation language should have.

3. The process (algorithm, control, problem solving) that the task uses. This provides a vocabulary for inference and control for the task.

The GT framework proposes that, as each task and its associated structure is identified, languages

[2]Each GT is a *strategy* from the viewpoint of the problem which it is helping to solve. It is a *task* from the viewpoint of the functionality that is to be achieved by the GT. It is in this sense that a GT is both a strategy for a task and a task in itself. This distinction is in fact much more general: e.g., diagnosis is a strategy in making patients feel better (i.e., one way to organize therapeutic actions is to base them on causes, finding which is, in fact, diagnosis), but it is a task which many expert systems are designed to solve.

be developed that encode both the problem solving strategy and knowledge that is appropriate for solving problems of that type. These languages facilitate KBS development by giving the knowledge engineer access to tools which work closer to the level of the problem, not the level of the implementation language such as rules or frames. However, for nontrivial problems it may be necessary to decompose it into subproblems such that each matches the functionality of some GT. For example, we will show how certain kinds of diagnostic problems can be decomposed into a number of GT's. This way of building complex KBS's also means that knowledge engineering environments should provide a *toolset* rather than a single tool.

This style of supporting higher level generic computational activities with appropriate constructs is well-known in computer science in general; high-level programming languages are attempts to provide the programmer with constructs for a variety of common functions. It is items 2 and 3 above, *viz., knowledge and inference*, that make the above specification different from the standard high-level language constructs in computer science and make it particularly appropriate for knowledge-based problem solving.

2 Some Generic Tasks and Their Specifications

While the approach has relevance to AI in general, as a practical matter, our work has concentrated on information-processing strategies useful for building systems for knowledge-rich problem solving, or so-called expert systems. Such systems emphasize the role of large amounts of domain knowledge compiled for specific problem solving tasks that characterize routine human expert behavior. Without intending any kind of completeness, we list here some of the tasks that we have found to be very useful in building practical knowledge-based systems. As we will see, a variety of diagnostic and routine design and planning problems come under this category. The description of the tasks in this section is necessarily cryptic and somewhat oversimplified, and is provided mainly for a quick overview and comparison. Hierarchical classification, hypothesis matching, and plan selection and refinement are described in some detail in later sections of the paper, and abductive assembly described in somewhat less detail. Please see the citations for details on the rest of the generic tasks.

Each description is organized by the function of the task, the tool in our GT toolset for the task, the knowledge and inference types that the tool supports, and other relevant annotations. In each case, the GT tool commits itself to *one* way of achieving the functionality. Also, in each case the control behavior is the default behavior, and should be thought of as describing a family of control types.

2.a Hierarchical Classification

Task specification

Input: given a situation description of in terms of features. Output: classify it, as specifically as possible, in a classification hierarchy. (Multiple classifications, where different classes characterize different parts of the situation description, are also possible.)

GT tool

CSRL [6] (Conceptual Structures Representation Language).

How CSRL works:

Forms of Knowledge

Classification hierarchy, access to knowledge that produce information about how well the data match the classificatory conceps (see Hypothesis Matching, below)

Inference and Control

(Simplified) *Establish* nodes, if successful *refine* the concept by considering children, if unsuccessful, *reject* node, and *prune* subtree.

Example use

Medical diagnosis can often be viewed as partly a classification problem [22]. Problems may be classified into types which may then suggest methods of solutions. The diagnostic portion of MYCIN [35] (see Clancey [18]) and PROSPECTOR [19] can be viewed as classification problem solving.

2.b Hypothesis Matching

Task specification

Input: given a concept (a hypothesis) and set of data (features) that describe the problem state, Output: decide how well the concept matches the situation. The task is a form of *recognition* [26].

GT Tool

HYPER (HYPothesis matchER) [24]

How HYPER works:

Forms of Knowledge

An hierarchy of evidence abstractions, lowest level at the level of data and the highest level at the level of the concept. Each node abstracts from its children into a higher level feature. In the particular version considered in our work, the abstractions are qualitative degrees of confidence. (See detailed description of HYPER, Section 3.c.3.)

Inference and Control

At each level a degree of confidence in the presence of the feature is computed from the features that constitute evidence for it, and this is performed recursively until a degree of confidence for the concept is computed. The basic theory is that recognition of a complex concept is performed by hierarchically computing intermediate abstractions from raw data.

Example use

Samuel's *signature tables* perform this kind of abstraction. Many forms of *recognition* can be performed by means of this strategy. For example, the concept may be a disease and the data may be patient data relevant to the disease, and we wish to know what the likelihood of the disease is. Bylander et al [8] discuss a class of strategies called *structured matching* (of which the HYPER strategy is a particular example) and show how ubiquitious it is in knowledge-based reasoning.

2.c Knowledge-Directed Information Passing

Task specification

Input: Given attributes of some data entities, Output: determine the attributes of other data of interest, but not directly known, but can be inferred from the available data.

GT tool

IDABLE (Intelligent DAta Base LanguagE).

How IDABLE works:

Forms of Knowledge

Data concepts organized as a frame hierarchy of types and subtypes of data objects in the domain. Data attributes are slots with default values, default methods of computation of values, or explicit procedural attachments which specify how to compute data attribute from related data.

Inference and Control

Data queries result in the default value being chosen if no information is available, or the knowledge-intensive procedures to be invoked for inference. The inference procedures themselves can be inherited from parent concepts as needed.

Example use

A diagnostic system may use a knowledge-directed database of this type for converting from sensor or chart values into data of direct relevance to diagnosis. Clancey's data absraction component of heuristic classification [18] can be achieved by this functionality.

2.d Synthesis by Plan Selection and Refinement

Task specification

Designing an object (device, program, plan) by hierarchical planning. Input: given specifications of the object to be designed Output: generate design of an object (device, program, plan) meeting the specifications.

GT tool

DSPL [3, 4] (Design Specialists and Plans Language).

How DSPL works:

Forms of Knowledge

Hierachical structure of the object to be designed known in advance (making it routine design). For each node in the hierarchy, precompiled design plans are known for making design choices. Failure handling knowledge available, and some parts of the plans are constraint satisfaction knowledge, i.e., knowledge of constraints to be met by the design parameters.

Inference and Control

Top down control is typically used. Design plans are chosen, choices made at that level of abstraction, and design *refined* by calling on plans for the children (i.e., components). Plan failures are passed up until failure handling knowledge is available to fix the design or choose alternate plan.

Example systems

The task performed by the expert system MOLGEN [20] and R1 [29] can be viewed in this way. A variant of this task has also been called *skeletal planning* in the literature.

2.e Abductive Hypothesis Assembly

Task specification

Input: Given a situation description and a set of hypotheses each explaining some aspects of the situation and each with some plausibility value Output: Construct a composite hypothesis that is the best explanation of the situation, i.e., explains all the data parsimoniously and as well as possible.

GT tool

PEIRCE (named after C. B. Peirce, who first described the form of inference known as abduction) [34].

How PEIRCE works:

Forms of Knowledge

Causal or other relationships between hypotheses (such as incompatibility, special case of), relative significance of data describing the situation.

Inference and Control

Assembly and crticism alternate. At each stage during assembly the problem solving is driven an attempt to explain the most significant datum remaining unexplained. The best hypothesis that offers to explain it is added to the composite hypothesis. During criticism, explanatorily superfluous parts are removed. This loops until all the data are explained or no hypotheses are left.

Example use

This task is a subtask in diagnostic reasoning as well as in theory formation in science. Portions of INTERNIST [30] and DENDRAL [5] systems perform this task.

2.f Some Implications of GT's

The following general points about GT's are worth noting at this point.

1. As mentioned, a number of well-known expert systems can be thought of as decomposable into subtasks, each of which corresponds to one of the above tasks. R1 performs a simplified type of plan selection and refinement. Mycin performs classification and data abstraction (one of the capabilities of our knowledge-directed information passing) in the diagnostic part, and plan selection (in the therapy part). Additional examples can be given. Note, however, that in all these instances, we are only pointing out that these systems perform these tasks, but not necessarily in the manner that we propose that the tasks be performed. Our claim will be that once we understand the knowledge requirements and the inference strageies for each of the tasks, we can use methods that are more natural for the tasks.

2. We have mentioned diagnosis a number of times in the above description as a problem that uses one or more of the above generic tasks, e.g., classification and hypothesis assembly. Note that, correspondingly, we do not have a generic task called diagnosis in the above list. The reason for this is that, while diagnosis is "generic" in the sense that the problem occurs in a number of domains and there are similarities in the methods that are domain-independent, it is still a "compound" task in the sense that a number of distinct types of

knowledge and inferences are used in the process of doing diagnosis. Thus the above list of tasks can be used as natural building blocks for putting together a diagnostic problem solver. (We shall soon describe how this can be done.) This illustrates an additional constraint in our sense of generic task. The task needs to have a coherence and simplicity to it in that it ought to be characterizable by a simple type of knowledge and a family of inference types. This is what makes them "building blocks."

3. Functional modularity is an important consequence of this point of view. As we have shown in a number of papers [7, 9, 10, 12, 15] this functional modularity makes system building and debugging easier, and the task-specific knowledge and control constructs help in knowledge acquisition and explanation.

4. The above list is not meant to be a complete list of generic tasks useful in knowledge-based problem solving. *In fact, quite a large part of AI -- from weak methods to qualitative* simulation to scripts and plans -- can be thought of as attempts to identify interesting problems, the kind of knowledge required for it and the kinds of inferences useful to perform them. Thus, in qualitative reasoning, the generic problem considered is one where given the structure of a system, the task is to derive the system's behavior in a qualitative way. The research program then identifies the knowledge and inference for the task. In the appropriate context, each of them can be thought of as possible generic tasks. Our goal in the development of the generic task theory and the tool set has been to produce a methodology and a technology that helps in the analysis, design, construction and debugging of practical knowledge systems and thus we concentrated on the generic tasks that we felt would be most useful at this stage in the development of the technology. Research is underway in our Laboratory on other such generic tasks, which would cover phenomena in deep models such as structure to behavior reasoning and functional reasoning.

In the next sections I will describe how certain kinds of diagnostic and design problems can be built in the GT framework.

3 Diagnostic Reasoning

3.a Information Processing in Diagnosis

Abstractly, diagnosis is the problem of finding a cause or set of causes that "best explain" a set of observations of a system, some of them indicating behavioral abnormality. In most nontrivial cases, the process is a form of *abductive* reasoning, i.e., the diagnostic conclusion is not deductively provable, but is the hypothesis that makes best sense taking all the information into account.

We can identify a class of systems that we call *compiled knowledge systems* for diagnosis. These systems have knowledge that is needed for diagnosis precompiled. This knowledge, at a minimum, would include:

- knowledge of possible causes in terms of which the diagnostic answer will need to be given.

- knowledge that helps map from observations to possible causes, i.e., evaluate how likely a

  given cause or subset of causes might be given the set of observations.

Many of the well-known diagnostic expert systems, e.g., Mycin [35], Internist [30], MDX [39], set-covering and Bayesian diagnostic systems have this knowledge compiled in the knowledge base. Where these systems differ is in the *form* this knowledge takes, in the way the actual inference processes work, and also in the control of reasoning. Such compiled knowledge systems concentrate their problem solving behavior only on the specific diagnostic problem at hand, rather than in activities that produce the needed knowledge. These systems ought to be contrasted with diagnostic systems which do not have the needed diagnostic knowledge in a readily usable form (or the diagnostic knowledge is incomplete), but must acquire them by other kinds of problem solving, e.g., by *deriving* it from structural models of the device under diagnosis, or from analysis of past diagnostic cases involving the device. See [13] for a discussion of the general issues surrounding the use of deep models and [38] for a discussion of how such model-based reasoning and compiled reasoning can be interlaced. The diagnostic architecture that I will be discussing in the following pages is a compiled knowledge architecture.

Diagnosis can be computationally complex: even with compiled knowledge, all subsets of hypotheses may in principle need to be evaluated and compared. This is mainly due to two reasons: one, hypotheses may interact, i.e., two hypotheses together may account for more or less observations than the union of the sets of observations that they explain individually; and, two different subsets may explain the same sets of data and principles of parsimony will need to be brought in to choose the better explanation. All diagnostic systems, be they formal, such as set-covering and Bayesian approaches, or "heuristic", such as Mycin, either squarely face this problem and end up with computationally intractable algorithms, or make more or less realistic assumptions about the domain that help them cut down the exhaustive search through the space of hypotheses combinations. (For example, its domain is such that Mycin can implicitly make assumptions of no interaction between diagnostic hypotheses.)

The GT architecture that we will propose shortly broadly decomposes the problem into a *classificatory* one, which generates highly plausible diagnostic hypotheses, which are then used by an *abductive assembly* component to produce the best composite hypothesis for the problem. The overall decomposition above brings significant computational advantages, since the assembly process now only needs to work with a much smaller number of initial hypotheses, with the option to seek out less plausible hypotheses as needed for explanatory completeness. This, in conjunction with the computational efficiencies that the proposed architectures for classification and abductive assembly individually possess, makes the above architecture computationally attractive whenever knowledge is available in appropriate forms: e.g., hierarchies for classification and explicit knowledge about causal and logical interactions among diagnostic hypotheses for the abductive assembly component.

3.b A GT Architecture for Diagnosis

The architecture has four components: *hierarchical classification, hypothesis matchers, abductive assembly*, and *knowledge-directed data abstraction and inference*. The hierarchical classifier navigates the space of malufunctions organized as one or more hierarchies. The hierarchical organization permits a quick determination of the plausible hypotheses with minimal search through the space of all possibile hypotheses. The result of the classification process is a small set of highly plausible hypotheses.

The classifier itself needs a mechanism to evaluate the degree of plausibility of each of the hypotheses. The knowledge necessary to evaluate the plausibility of a classificatory hypothesis can be localized to each hypothesis in the context of the hierarchy. This can be done by a *hypothesis matching* component which evaluates any given hypothesis in the classification hierarchy by matching the data with expectations for the concept and which outputs a qualitative degree of confidence in the hypothesis (and the observations the hypothesis can explain). Thus the classifier, in conjunction with the hypothesis matchers for each of the concepts, can output plausible diagnostic hypotheses with the data it can explain attached to each of the hypotheses.

The output of the classifier goes to an *abductive hypothesis assembly* component, which puts together a subset of these hypotheses as a composite hypothesis that best explains the data. This process must consider the interactions that occur among the causes that correspond to the hypotheses in order to ensure internal coherence among combinations of hypotheses. The knowledge concerning hypothesis interactions can be explicitly represented for each hypothesis, thus being consulted only if that hypothesis is included in the composite hypothesis. This process must also assemble a diagnosis which meets various criteria of parsimony, completeness and plausibility. In the more complex uses of this architecture, the classification hierarchy may be asked to refine originally less plausible hypotheses if the explanatory power of the best composite hypothesis so far assembled is insufficient to cover all the observations that need explanation.

In many diagnostic problems the level of abstraction of the data which are available may be different from that required for the concept matcher, or additional inferences from available data may be needed to generate data that the diagnostic concept matcher can recognize as relevant. A necessary addition to this architecture is a database which uses domain knowledge to make the inferences and abstractions. In the proposed architecture, the hypothesis matcher can communicate with a system for *data retrieval/abstraction/inference*, whose task is a form of *knowledge-directed information passing* component which can convert data at to low a level of abstraction into diagnostically significant data.

The important point is that each of the modules above is generic:

- Each is a strategy independent of diagnosis and can be used in a number of other high level tasks. The abductive assembler, e.g., can just easily accept input from a plan recognizer so that it assembles a best explanation of various sightings in a battle situation. The data abstractor can be just easily used by a therapy planner, and so on.

- Each strategy, as we shall see, uses characteristic knowledge and inference, maing it possible to focus the problem solving effort in a manner appropriate for the task.

The functionality of Clancey's *heuristic classification*, consisting of the subtasks of *data abstraction*, *heuristic matching*, and *refinement* can be achieved by three modules in our architecture: the hypothesis-matcher performs the heuristic matching task, the database component performs, among others, the data abstraction task, and the classifier performs the refinement task. The architecture has the additional capability of handling multiple malfunctions because of the abductive assembly component.

In sum the task of diagnosis can often be handled by the building blocks in the architecture diagramed in Figure 1. Thus, if the appropriate knowledge is available, a compiled knowledge diagnostic system can be built using the CSRL, HYPER, PEIRCE and IDABLE tools.

Gomez and Chandrasekaran [22] pointed out the importance of classification for diagnosis and Mittal [31] described the architecture of MDX, which included all the components except abductive assembly. Josephson et al [27] added abductive assembly as part of a general architecture for abduction.

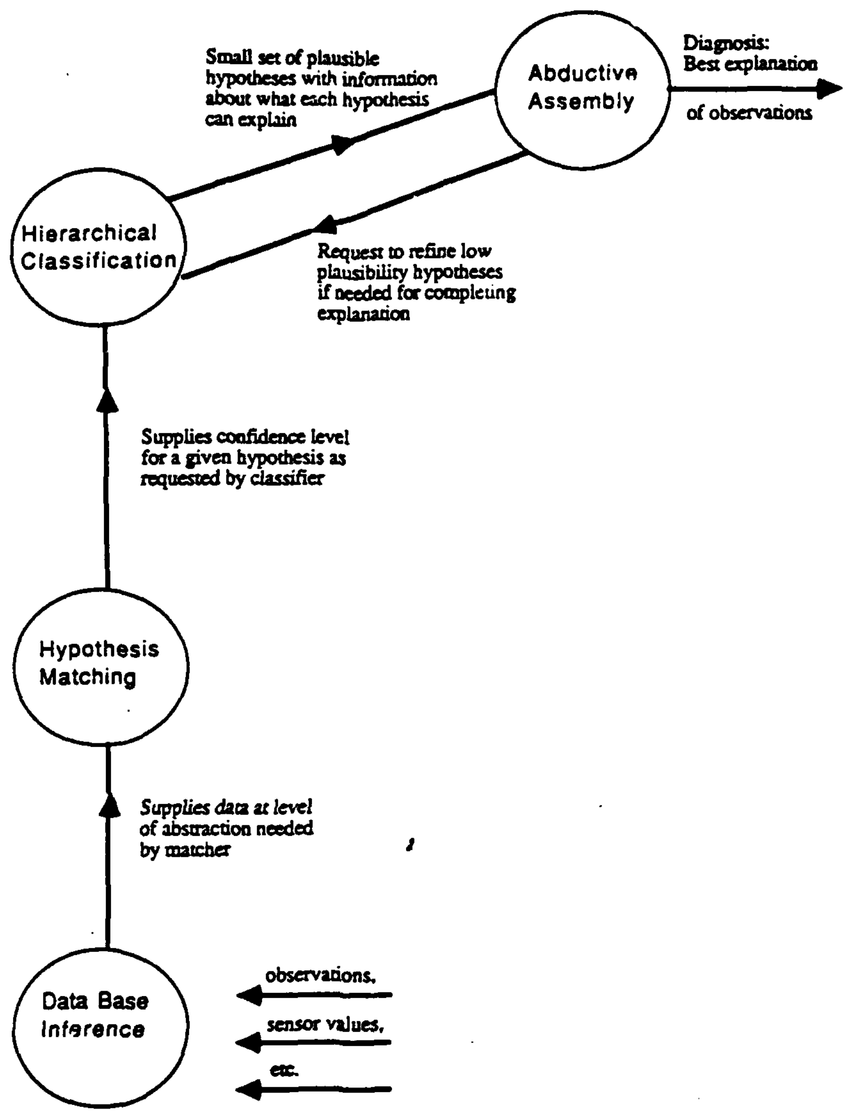

Small set of plausible hypotheses with information about what each hypothesis can explain
Abductive Assembly
Diagnosis: Best explanation of observations
Hierarchical Classification
Request to refine low plausibility hypotheses if needed for completing explanation
Supplies confidence level for a given hypothesis as requested by classifier
Hypothesis Matching
Supplies data at level of abstraction needed by matcher
Data Base Inference
observations,
sensor values,
etc.

Figure 1: *A Generic Task Architecture for Diagnosis With Compiled Knowledge*

Describing how all the components of the diagnostic architecture can be built and integrated requires much more space than we have available. We describe in some detail the use of CSRL (the version to be described has HYPER embedded in it) for classification and hypothesis matching, and describe the assembly process in less detail. References [25] and [34] give further information on the tools Idable and Peirce.

3.c Classificatory Problem Solving and CSRL[3]

[3]Much of the material in this section is from B. Chandrasekaran and William F. Punch III, "Hierarchical classification: its usefulness for diagnosis and sensor validation," invited talk at the Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program, March 9-11, 1987.

3.c.1 Classificatory Hierarchies

Hierarchical classification (HC) is a particular method of performing the classification task. HC requires the availability of a classification hierarchy that organizes the classificatory hypotheses. Medical diagnosis, e.g., uses disease hierarchies, and in many engineering domains, malfunction hierarchies are quite common.

In Section 2, we gave a characterization of hierarchical classification. Figure 2 illustrates a fragment of a tree from a hierarchical classification system for the diagnosis of Fuel System malfunctions in a car engine.

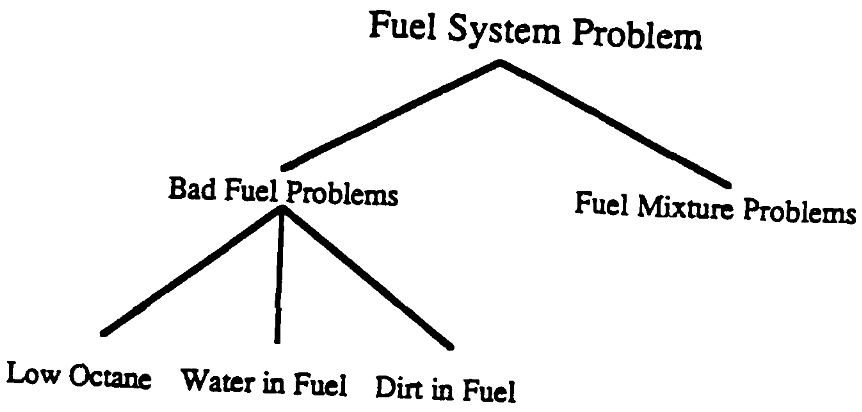


Figure 2: *Fragment of Fuel System classification tree. In this case, the hierarchy is largely of classes and subclasses of "causes." In other cases, the subclasses may be subfunctions, or physical parts.*

Note that as the hierarchy is traversed from the top down, the categories (or in this particular case, hypotheses about the failure of the fuel system) become more specific. Thus the children of the hypothesis Bad Fuel Problems can be broken into more specific hypotheses of Low Octane, Water in Fuel and Dirt in Fuel.

Each node in the hierarchy is responsible for calculating the "degree of fit" or *confidence value* of the hypotheses that the node represents. For example, the Bad Fuel Problems node is responsible for determining if there is a bad fuel problem and the degree of confidence it has in that decision. Each node can be thought of as a "specialist" in determining if the hypothesis it represents is present. To create each specialist, knowledge must be provided to make this confidence value decision. The general idea is that each specialist specifies a list of *features* that are important in determining whether the hypothesis it represents is present and a list of *patterns* that map combinations of features to confidence values. In the Fuel System Problems specialist, such features might include gas mileage problems, poor performance, difficulty in starting the engine etc. One pattern might be that if all the features are present, then the Fuel System Problems hypothesis is likely.

3.c.2 The Control Strategy of Hierarchical Classification

Given that the knowledge of the system is organized as a set of specialists in a hierarchy, how can the hierarchy be *efficiently* traversed? This process is primarily accomplished through a type of hypothesis refinement called *establish-refine.* Simply put, a specialist that *establishes* its hypothesis (has a high confidence value) *refines* itself by activating its more detailed sub-specialists. A hypothesis that is ruled out or rejected its hypothesis (has a low confidence value) is not refined, thus effectively pruning the subtree below it. The reason for this becomes obvious when one thinks again of how the specialists are organized. The subhypotheses of Fuel System Problems, for example, are simply more detailed hypotheses. If there is no evidence for Fuel System Problems (it is ruled out), then there is no point in examining more detailed hypothesis about failures of the fuel system.

The process of establish-refine continues until no more refinements can take place. This can occur either by reaching the tip level hypotheses of the hierarchy or by ruling out mid-hierarchy hypotheses.

3.c.3 CSRL, a Language Tool for Hierarchical Classification Systems

CSRL (Conceptual Structure Representation Language) [6] is a language for writing hierarchical classification expert systems. The current version of CSRL is really a mixture of the shells of both hierarchical classification and hypothesis matching. A new version of the hypothesis matcher shell is available as HYPER. In this section, we will describe the older form of CSRL.

CSRL allows a knowledge engineer to do three things:

1. Create a hierarchy of malfunction hypotheses in a particular domain.

2. Encode the pattern matching knowledge for each hypothesis into a specialist.

3. Control the process of establish-refine problem solving.

Encoding the Hierarchy of Malfunctions

In CSRL, a hierarchical classification system is implemented by individually defining a specialist for each malfunction hypothesis. The super- and sub-specialists of a specialist are declared within the definition. Figure 3 is a skeleton of a specialist definition for the Bad Fuel node from Figure 2. The declare section specifies its relationships to other specialists. The other sections of the specialist will be examined later.

```
(SPECIALIST BadFuel
  (DECLARE (SUPERSPECIALIST FuelSystem)
        (SUBSPECIALIST LowOctane WaterInFuel
         DirtInFuel))
  (KGS ...)
  (MESSAGES ...))
```

Figure 3: *Skeleton specialist for* BadFuel. *The code specifies the location of* BadFuel *in the hierarchy, points to the knowledge groups that contain information about how to establish or reject the concept, and contains messages that specify control behavior.*

Designing a classification hierarchy is an important part of building a CSRL expert system, but the exact structure of the final system is a pragmatic decision rather than a search for the perfect hierarchy. The main criterion for evaluating a classification hierarchy is whether enough evidence is normally available to make confident decisions. To decompose a specialist into its subspecialists, the simplest method is to ask the domain expert what subhypotheses should be considered next. The subhypotheses should be subtypes of the specialist's hypothesis, and will usually differ from one another based on a single attribute (e.g., location, cause).

For the diagnosis problem the criteria for forming classification hierarchies are discussed, with examples from the medical domain, by [7], and in the engineering domain by [32]. The hierarchy may mix function-subfunction and part-subpart views depending upon the way diagnostic reasoning actually works in the domain. Multiple hierarchies of the same domain, each from a different perspective, are also useful for some domains: the MDX-2 system of Sticklen [39] uses such multiple hierarchies.

Encoding Pattern-Match Knowledge

The knowledge groups in the kgs section contain knowledge that matches the features of a specialist against the case data. Each knowledge group is used to determine a confidence value for some subset of features used by the specialist. As such, a knowledge group becomes an abstraction of evidence, representing an *evidential abstraction* of a particular set of features important to establishing the specialist. A knowledge group is implemented as a cluster of production rules that maps the values of a list of expressions (boolean and arithmetic operations on data, values of other knowledge groups) to some conclusion on a discrete, symbolic scale.

As an example, Figure 4 is the relevant knowledge group of the BadFuel specialist mentioned above. It determines whether the symptoms of the automobile are consistent with bad fuel problems. The expressions in the MATCH part queries the user (who acts as the database for this case) concerning whether the car is slow to respond, starts hard, has knocking or pinging sounds, or has the problem when accelerating. ASKYNU? is a LISP function which asks the user for a Y, N, or U (unknown) answer from the user, and translates the answer into T, F, or U, the values of CSRL's three-valued logic (Note that any LISP function may be used here). The results of the MATCH expressions are then compared to a condition list in the WITH part of the knowledge group. For example, the first pattern "T ? ?" in the figure tests whether the first match expression (ASKYNU? "Is the car slow to respond") is true (the ? means doesn't matter). If so, then -3 becomes the value of the knowledge group[4]. Otherwise, subsequent patterns "? T ?" or "? ? T" are evaluated. The value of the knowledge group will be 1 if no rule matches. This knowledge group encodes the following matching knowledge:

> **If the car is slow to respond or if the car starts hard, then BadFuel is not relevant in this case. Otherwise, if there are knocking or pinging sounds and if the problem occurs while accelerating, then BadFuel is highly relevant. In all other cases, BadFuel is only mildly relevant.**

```
(RELEVANT TABLE
 (MATCH
   (ASKYNU? "Is the car slow to respond")
   (ASKYNU? "Does the car start hard")
   (AND (ASKYNU? "Do you hear knocking or
              pinging sounds")
      (ASKYNU? "Does the problem occur while
              accelerating"))
  WITH (IF T ? ?
      THEN -3
      ELSEIF ? T ?
      THEN -3
      ELSEIF ? ? T
      THEN 3
      ELSE 1)))
```

Figure 4: "Relevant" *knowledge group of* BadFuel. *The ASK arguments are questions to the user, but they can also be queries to the database. The argument of WITH specify truth tables in the knowledge group.*

Figure 5 is the summary knowledge group of BadFuel. Its MATCH expressions are the values of the relevant and gas knowledge group (the latter queries the user about the temporal relationship between the onset of the problem and when gas was last bought). In this case, if the value of the

[4]In this case, the values assigned are on a discrete scale from -3 to 3, -3 representing ruled-out and 3 representing confirmed.

relevant knowledge group is 3 and the value of the gas knowledge group is greater then or equal to 0, then the value of the summary knowledge group (and consequently the confidence value of BadFuel) is 3, indicating that a bad fuel problem is very likely.

```
(SUMMARY TABLE
  (MATCH RELEVANT gas
   WITH (IF 3 (GE 0)
      THEN 3
      ELSEIF 1 (GE 0)
      THEN 2
      ELSEIF ? (LT 0)
      THEN -3)))
```

Figure 5: "Summary" *knowledge group of* BadFuel.

This method of evidence combination allows the calculation of the confidence value to be hierarchically organized. That is, the results of any number of knowledge groups can be further abstracted by a knowledge group that can combine their values into a single confidence value.

As mentioned earlier the above pattern matching knowledge and problem solving structure is a generic task that we have identified as hypothesis matching, and a separate shell called HYPER is available to capture just this functionality.

The mapping from data to confidence in a concept is a form of probabilistic mapping. The symbolic degrees of confidence are qualitative measures of subjective likelihood. However, the way data combine to produce a confidence for a higher level feature is *not* modeled by any normative calculus, be it Bayesian or one based on fuzzy sets, but *directly* obtained from the domain expertise localized to that particular context. There are important issues of how this view of handling uncertainty differs from the more traditional formal methods for which we refer the reader to [11, 6].

Encoding of Establish-Refine Strategy

The MESSAGES section of a specialist contains a list of message procedures which specify how the specialist will respond to different messages from its superspecialist. ESTABLISH and REFINE are the predefined messages in CSRL though others may be created by the user. The establish message procedure of a specialist determines the *confidence value* (i.e., the degree of fit) of the specialist's hypothesis. Figure 6 illustrates the establish message procedure of the BadFuel specialist. relevant ano summary are names of knowledge groups of BadFuel (see previous section). SELF is a keyword which refers to the name of the specialist. This procedure first tests the value of the relevant knowledge group. (If this knowledge group has not already been evaluated, it is automatically evaluated at this point.) If it is greater than or equal to 0, then BadFuel's confidence value is set to the value of the summary knowledge group, else it is set to the value of the relevant knowledge group. A value of +2 or +3 indicates that the specialist is established. In this case, the procedure corresponds to the following strategy.

First perform a preliminary check to make sure that BadFuel is a relevant hypothesis to hold. If it is not (the relevant knowledge group is less than 0), then set BadFuel's confidence value to the degree of relevance. Otherwise, perform more complicated reasoning (the summary knowledge group combines the values of other knowledge groups) to determine BadFuel's confidence value.

The refine message procedure determines what subspecialists should be invoked and the messages they are sent. Figure 7 shows a refine procedure which is a simplified version of the one that BadFuel uses. SUBSPECIALISTS is a keyword which refers to the subspecialists of the current specialist. The procedure calls each subspecialist with an ESTABLISH message. If the subspecialist establishes itself (+? tests if the confidence value is +2 or +3), then it is sent a REFINE message.

```
(ESTABLISH (IF (GE relevant 0)
      THEN (SETCONFIDENCE self summary)
        else (SETCONFIDENCE self relevant)))
```

Figure 6: *Establish procedure of* BadFuel. *If it has been evaluated not to be relevant, that sets the confidence value to be negative. If it is relevant then more complex matching is invoked to set the confidence value.*

```
(REFINE (FOR specialist IN subspecialists
        DO (CALL specialist WITH ESTABLISH)
          (IF (+? specialist)
            THEN (CALL specialist
                WITH REFINE))))
```

Figure 7: *Example refine procedure. This specifies the control behavior for exploring the successors of a classificatory hypothesis that has been established.*

3.c.4 The Computational Advantages of Hierarchical Classification

The major advantage of a hierarchical classification system is the organization of both the hierarchy of malfunctions and the knowledge groups within a specialist. This organization allows an efficient examination of the knowledge of the system based on need.

Consider again the hierarchy of Figure 2. The problem solving begins by evaluation of the specialist Fuel System Problems. If that specialist establishes, then the two sub-specialists Bad Fuel Problems and Fuel Mixture Problems are invoked. If however, the Bad Fuel Specialist does not establish, then none of its sub-specialists will be invoked. Thus, if a specialist rules out (i.e does not establish), then none of the knowledge of the sub-specialists need be run.

The same is true of the knowledge groups in the specialist. Only that knowledge necessary to confirm or deny the knowledge group is run. If a row of the knowledge group matches, then none of the subsequent rows are evaluated. Again, this results in running only the knowledge necessary for the problem at hand.

Compare this with other hierarchical approaches to diagnosis. The *fault tree* is a sequence of causally related events that leads to an observable symptom in the system. Given an initial malfunction, all possible causal results of the event are traced out, terminating with the symptoms that would be observed by a human diagnostician. When applied to an entire system, the result is a network of events that represent all the causal relationships of the system's constituent parts. While useful in design tasks, application of fault trees to diagnosis has a number of problems.

1. The combinatorial fan-out from an initial event can be very large. This makes the job of creating and traversing the network difficult. Compare this with the abstraction of hypotheses in hierarchical classification systems. Each node in the hierarchy represents a malfunction hypothesis that is listed in more detail through its sub-specialists. If many sub-specialists occur in the hierarchical decomposition of the domain, more levels of

abstraction can be introduced to limit the fan-out. Such abstraction does not exist in fault tree representations.

2. Fault trees make no attempt to limit the number of nodes of the network that must be evaluated. Given a significant event, all possibilities are examined. However, hierarchical classifiers make use of the abstraction of malfunction hypotheses to limit the number of nodes that must be examined based on the data of the case.

The issues of control and communication in hierarchical classification can be more complex than our description in this paper. Gomez and Chandrasekaran [22] describe the use of blackboards in exchanging information between different portions of the hierarchy. Sticklen, et al [38] describe the more complex control issues that need to be faced in some situations, and Sticklen [39] describes the use of multiple hierarchies in classification.

3.d Hypothesis Assembly in Diagnosis

In typical diagnosis problems, the available data cannot always be explained by one malfunction hypothesis, but may require several hypotheses which must be combined in order to account for the observations. As the space of hypotheses grow in size, the problem of finding the best combination of hypotheses which apply to a particular situation becomes exponentially more difficult. Hierarchical classification can trim down the space of applicable hypotheses tremendously, yet it may still be necessary to find a subset of the hierarchy's plausible candidates which gives the simplest and best diagnosis.

For example, suppose that the Auto-Mech hierarchy were extended to classify malfunctions for other car subsystems, including the braking mechanism. When classifying a case which has data such as "The car won't start when cold," "The engine runs roughly," "The brakes are hard to push," and "The car doesn't stop quickly," several diagnostic hypotheses may be found to be applicable, such as 'water in fuel' and 'loss of brake fluid', among others. In this case, the classification hierarchy has trimmed the set of all fuel and brake system diagnostic hypotheses down to a handful of highly relevant ones. However, the classification mechanism as such is incapable of determining whether this subset of relevant hypotheses accounts for all of the data. Furthermore, some of these hypotheses may be inconsistent or superfluous with respect to each other. Therefore, it is necessary to invoke a process of hypothesis assembly to complete the diagnosis by assembling a parsimonious subset of these hypotheses which gives the best explanation. In this case, the final subset of hypotheses could be 'water in fuel' and 'loss of brake fluid', since both are highly plausible, between the two of them they account for all the data, and neither is inconsistent or superfluous with regard to the other.

To carry this example further, it is possible that the hypothesis assembler may uncover relevant relationships which are only implicitly represented in the classification hierarchy. For example, in some models of cars the vacuum generated by airflow through the engine is diverted to assist the braking mechanism. Thus a punctured vacuum hose reduces the airflow through the carburetor, causing the engine to run roughly, and disabling the assistance to the brakes. When running the Auto-Mech hierarchy on the previous case, for an appropriate model of car, the following hypotheses may be found to be relevant: 'engine vacuum hose punctured' and 'brake vacuum assist inoperational'. Both of these hypotheses need to be present in the classification hierarchy, because the first refines the 'engine problem' hypothesis, whereas the second refines the 'brake system problem' hypothesis. In this case, however, it is clear that the second problem is causally related to the first. Therefore, when the hypothesis assembler is putting together the best explanation for hypothesis assembler is putting together the best explanation for the data, it will uncover that 'engine vacuum hose punctured' accounts for all the data that 'brake bacuum assist inoperational' accounts for, and propose 'engine vacuum hose punctured' as a one-hypothesis set which best explains the data.

Combining the process of hierarchical classification and hypothesis assembly in this way results in an efficient process for diagnosis. Classification without assembly cannot evaluate the final diagnosis in the case of multiple failures. Assembly without classificaiton is an intractable problem in the general case. The issue of complesity in hypothesis assembly is particularly noticeable in the medical domain, where physiological and anatomical subsystems interact in highly complex ways, When both generic tasks are combined, however, their interaction reduces the complexity of the diagnostic problem while maintaining the ability to find the best explanation when there is one.

Josephson, et al [27] is a good source of how the abductive assembly process works. Punch et al [34] describe the tool PEIRCE, which is intended to build abductive assembly systems.

3.e Applications of the Abductive Architecture

A number of diagnostic systems have been built using the hierarchical classification approach provided by the CSRL tool. This section enumerates some the these applications and their domains.

It should be noted that of the following systems, Auto-Mech is strictly a pedagogical system, the Nuclear Power and Chemical Engineering systems are initial explorations for yet to be developed systems and Red, WELDEX and ROMAD are being developed to be used in real world situations.

Auto-Mech [40]

Auto-Mech is an expert system which diagnoses fuel problems in automobile engines. The purpose of the fuel system is to deliver a mixture of fuel and air to the air cylinders of the engine. It can be divided into major subsystems (fuel delivery, air intake, carbuertor, vacuum manifold) which correspond to initial hypotheses about fuel system faults.

Auto-Mech consists of 34 CSRL specialists in a hierarchy which varies from four to six levels deep. Before running, Auto-Mech collects some initial data from the user. This includes the major symptom that the user notices (such as stalling) and the situation when this occurs (e.g., accelerating and cold engine temperature). Any additional questions are asked while Auto-Mech's specialists are running. The diagnosis continues until the user is satisfied that the diagnosis is complete.

A major part of Auto-Mech's development was determining the assumptions that would be made about the design of the automobile engine and the data that the program would use. Different automobile engine designs have a significant effect on the hypotheses that are considered. A carbureted engine, for example, will have a different set of problems than a fuel injected engine (the former can have a broken carburetor). The data was assumed to come from commonly available resources. The variety of computer analysis information that is available to mechanics today was not considered in order to simplify building Auto-Mech.

Red [37]

Red is an expert system whose domain is red blood cell antibody identification. An everyday problem that a blood bank contends with is the selection of units of blood for transfusion during major surgery.. The primary difficulty is that antibodies in the patient's blood may attack the transfused blood, rendering the new blood useless as well as presenting additional danger to the patient. Thus identifying the patient's antibodies and selecting blood which will not react with them is a critical task for nearly all red blood transfusions.

The Red expert system is composed of three major subsystems, one of which is implemented in CSRL. The non-CSRL subsystems are a data base which maintains and answers questions about reaction records (reactions of the patient's blood in selected blood samples under a variety of conditions), and a overview system, which assembles a composite hypothesis of the antibodies that would best explain the reaction record. (This assembly is itself a generic task called "abductive assembly" and a tool called PEIRCE can be used to build the assembly system.) CSRL is used to implement specialists corresponding to the common blood antibodies and to each antibody subtype (different ways that the antibody can react).

The major function of the specialists is to rule out antibodies and their subtypes whenever possible, thus simplifying the job of the overview subsystem, and to assign confidence values, informing overview of which antibodies appear to be more plausible. The specialists query the data base for information about the lab test results and other patient information, and also tell the data base to perform certain operations on reaction records.

Complex Mechanical Systems

CSRL has been used in creating expert systems that do diagnosis of faults both in the domain of Nuclear Power Plants and in the domain of Chemical Engineering.

The Nuclear Power Industry must be very careful in the maintenance of running power plants since mistakes can prove costly not only in terms of power plant damage but also in terms of radiation leakage and broad environmental damage. Nuclear Power Plants are therefore heavily monitored in many areas, so heavily in fact that it is difficult (if not impossible) for the operator to maintain an understanding of just what exactly is going on. The Nuclear Power Plant expert system [23] is designed to take in large amounts of data and classify them into one of approximately 25 different failures. One advantage of the CSRL approach is that the operator can be informed of a high level view of the problem if no specific failure can be discovered.

The problems of the Chemical Engineering Plant are similar, but it does have a number of differences. While safety is also of concern, there is also the problem of product quality in a Chemical Engineering Plant. If a malfunction occurs that produces an unusable product, the operation must be brought quickly back into line or large amounts of material will be wasted. The Chemical Engineering expert system [36] does diagnosis of a typical reactor producing a solid product as a result of the reaction of liquid product and oxygen. It consists of approximately 30 specialists that represent hypotheses about failures of the various physical parts of the plant. In addition to data that monitors the state of the reactor, these specialists also use data about product quality to make the confidence value decision.

Other Real World Uses of CSRL

CSRL is being used to develop two commercial systems by the Knowledge Based Systems group at the Battelle Columbus Institute. WELDEX and ROMAD are diagnostic systems for, respectively, detecting welding defects and evaluating machinery. A brief description of WELDEX follows.

WELDEX identifies possible defects in a weld from radiographic data of the weld. Industry standards and regulations require careful inspection of the entire weld and a very high level of quality control. Thus for industries which rely on welding technology, such as the gas pipeline industry, radiograph inspection is a tedious, time-consuming, and expensive part of their operations.

This problem can be decomposed into two tasks: visual processing of the radiograph to extract relevant features of the weld, and mapping these visual features to the welding defects which give rise to them. WELDEX is intended to perform the second task. The current prototype consists of 25 CSRL specialists that are organized around different regions of the weld, taking advantage of the fact that each class of defects tends to occur in a particular region. The knowledge groups in these specialists concentrate on optical contrast, shape, size and location of the radiograph features. A customer version of WELDEX is currently being developed. Future work is anticipated on developing a visual processing system whose output would be processed by WELDEX, thus automating both parts of the radiograph inspection problem.

4 Routine Design and DSPL[5]

4.a Subprocesses in Design

Design is in general complex and, from the viewpoint of AI, a relatively poorly understood activity. For our purposes here, it is useful to think of design problem solving as having two sets of parts: those that "generate," i.e., propose designs or parts of designs and those that "test," i.e., analyze, critique and evaluate designs. Evaluating a design may involve problem solving behavior such as qualitative simulation (e.g., to see if the projection from the wheel will rub against the body during rotation) or quantitative analyses (e.g., finite element methods to evaluate stress in a design component to see if the maximum stress is below the specifications), but these processes are not specific to design as a problem solving activity. On the other hand, the processes that participate in the "generate" portion are specific to design. In [16] we have identified some of these generic processes:

- design problem decomposition,
- design plan instantiation and expansion,
- retrieval and modification of similar designs and,
- global satisfaction of constraint equations.

Among the above four processes, the first two are especially important for the discussion in this paper. In design by decomposition pre-stored domain knowledge is available which proposes possible decompositions of the design problem into specific subproblems, each hopefully of lesser complexity. In design by plan selection, instantiation and refinement, similarly a prestored *design plan* is available setting out a set of steps, some of them involving making some design commitments and others possibly involving calling other design plans, for solving the design problem at hand. The combination decomposition and design plan instantiation and refinement can lead to quite complex problem solving. In design from past cases, both the decomposition and design plan are implicitly available for a previous design case, but they typically call for further criticism and modification.

4.b Classes of Design

The framework suggests that design by decomposition (i.e., breaking problems into subproblems), by plan selection, and by plan synthesis (as a last resort) are the core processes in knowledge-based design. This suggests an informal classification of design problems based on the difficulty of these subtasks or processes.

4.b.1 Class 1 Design

This is open-ended "creative" design. Goals are ill-specified, and there is no storehouse of effective decompositions, not to speak of design plans for subproblems. Even when decomposition knowledge is available, most of the effort is in searching for potentially useful problem decompositions. For each potential subproblem, further work has to be done in evaluating if a design plan can be constructed. This design problem is not routine. The average designer in industry will rarely, if ever, do Class 1 design: such design leads to an invention or new products.

[5]Parts of this section taken from a number of joint papers by the author with D. C. Brown.

4.b.2 Class 2 Design

Class 2 design is characterized by powerful problem decompositions already available, but design plans for some of the component problems in need of *de novo* construction or substantial modification. Design of a new automobile, e.g., does not involve new discoveries about decomposition: the structure of the automobile has been fixed for quite a long time. On the other hand, several of the components in it constantly undergo major technological changes, and routine methods of design for some of them may no longer be applicable.

Complexity of failure analysis will also take a problem away from routine design. Even if design plans are available, if the problem solver has to engage in very complex problem solving procedures in order to decide how to backtrack, the advantage of routine design is reduced. In short, whenever substantial modifications of design plans for components are called for, or when synthesis in the design plan space is especially complicated, we have a Class 2 problem.

4.b.3 Class 3 Design

This is relatively routine design: effective problem decompositions are known, compiled design plans for the component problems are known, and actions to take on failure of design solutions are also explicitly known. There is very little complex auxiliary problem solving needed. In spite of all this simplicity, the design task itself is not trivial: complex backtracking can still take place. The design task is still too complex for simple algorithmic solutions or table look up.

Class 3 problems are routine design problems, but still requiring knowledge-based problem solving. The ensuing sections of this paper deal with an approach to building knowledge-based systems for routine design problems of this type. The processes described here can work in conjunction with auxiliary problem solvers of various types, but we do not discuss such additional problem solvers here. The examples used all assume that the information to be provided by the auxiliary design processes, e.g., design criticism, verification, and subproblem constraint generation, are all available in a compiled manner.

4.b.4 A Class 3 Product

In a large number of industries, products are tailored to the installation site while retaining the same structure and general properties. For example, an Air-cylinder intended for accurate and reliable backward and forward movement of some component will need to be redesigned for every new customer in order to take into account the particular sp[illegible] into which it must fit or the intended operating temperatures and pressures. This is a design task, but a relatively unrewarding one, as the designer knows at each stage of the design what the options are and in which order to select them. Note that that doesn't mean that the designer knows the complete sequence of steps in time (i.e., the trace) in advance, as the designer has to be in the problem-solving situation before each decision can be made. There are just too many combinations of requirements and design situations to allow an algorithm to be written to do the job. This class of problems, while simple, is not by any means trivial: in fact, a typical class 3 problem involves more complex problem solving behavior at the design level than say is implicit in R1 [29].

DSPL is a language designed by D. C. Brown [3] which captures the problem decomposition knowledge in the form of a design hierarchy of *design specialists* and the planning knowledge of each specialist is in the form of *design plans*. The specialists also have a certain amount of compiled *failure handling knowledge*, i.e., knowledge to help them recover when any of the chosen plans fail to accomplish their mission.

The approach taken is to consider design knowledge to be in the form of active cooperating design specialists. These specialists are organized in a hierarchy that reflects the human designer's conceptual organization of the design activity. Specialists use their own local design knowledge, but can also use the specialists directly below them in the hierarchy. This use is controlled by plans embedded in every specialist. Each specialist is responsible for some portion of the design, while it plans represent alternative methods for designing that portion. Communication between specialists is in the form of messages that flow up and down the hierarchy between the specialists and between their local agents (i.e., local design knowledge)[6]. Messages flowing up may indicate failure or success.

The domain chosen was that of designing an Air Cylinder that was used by a local company in many pieces of equipment but which needed to be designed again each time due to changing requirements, such as the air pressure and the length of the stroke. A system called AIR-CYL has been written that does the design given a set of requirements.

The rest of this section will describe DSPL using the example of AIR-CYL [4].

4.c DSPL : The Design Specialists and Plans Language

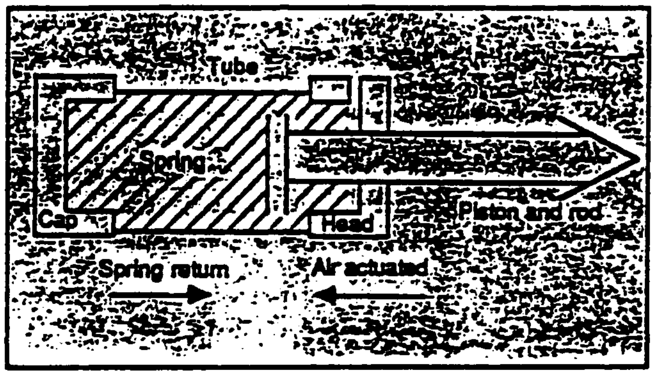


Figure 8: *An air-cylinder*

The air cylinder (AC) has about 15 parts, almost all of which are manufactured by the company according to their own designs, as their requirements are such that the components cannot be purchased. The AC is redesigned and changed slightly for applications with markedly different requirements. This characteristic makes it routine design in type. In operation, compressed air forces a piston back into a tube against a spring. Movement is limited by a bumper. The spring returns the piston, and the attached "load," to its original position when the air pressure drops.

The corresponding design specialist hierarchy is given in Figure 9. The expert system for design of aircylinders is organized as a hierarchy of such specialists.

DSPL provides a way of writing declarations of Specialists, Plans, Tasks, Steps, Constraints, Failure Handlers, Redesigners, Sponsors and Selectors, allowing the user to specify the knowledge contained in them. In the following section we will address each of these declarations in turn.

[6]We use the term "agent" to denote any chunk of knowledge that performs some action with well-defined functionality to it. The term "specialist" refers to those agents which correspond to the nodes in the part-subpart hierarchy of the object under design. Each specialist consists of further agents which take care of the subtasks of the specialists. In DSPL, these agents come in a number of different types.

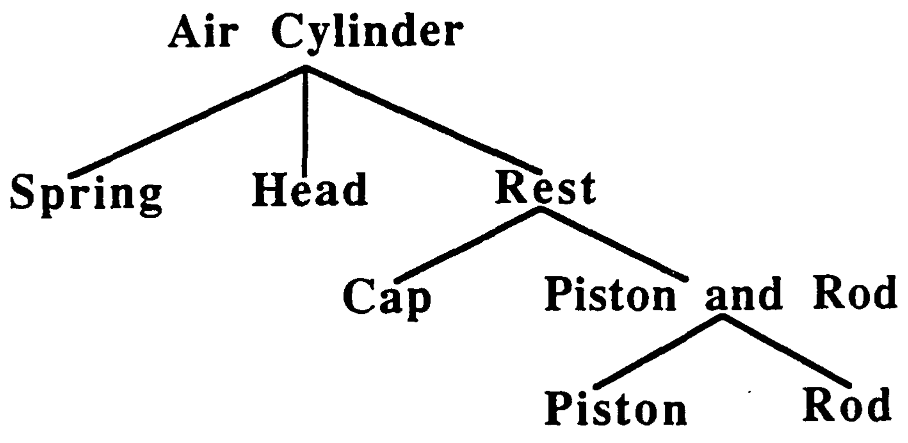


Figure 9: *Specialist hierarchy for designing air cylinder. Design hierarchies may follow a function-subfunction or a part-subpart or a combination thereof as principles for organizing the hierarchy.*

To build a Design Expert System (DES), the user declares in DSPL all the agents (i.e., active design knowledge) required, and then allows the underlying system to link them together after some checking. Once formed the DES can be invoked by requesting a design from the top-most specialist. The design then proceeds according to the specialist's plans. After a successful termination the design data-base contains the completed design. If failure occurs reasons are given. The DSPL system provides the underlying problem-solving control.

4.c.5 Design Agents
Specialists

A Specialist is a design agent that will attempt to design a section of the component. The specialists chosen, their responsibilities, and their hierarchical organization will reflect the mechanical designer's underlying conceptual structure of the problem domain. Exactly what each specialist's responsibilities are depends on where in the hierarchy it is placed. Higher specialists have more general responsibilities. The top-most specialist is responsible for the whole design. A specialist lower down in the hierarchy will be making detailed decisions. Each specialist has the ability to make design decisions about the part, parts or function in which it specializes. Those decisions are made in the context of previous design decisions made by other specialists. A specialist can do its piece of design by itself, or can utilize the services of other specialists below it in the hierarchy. We refer to this cooperative design activity of the specialists as Design Refinement.

Every specialist also has some local design knowledge expressed in the form of constraints. These will be used to decide on the suitability of incoming requirements and data, and on the ultimate success of the specialist itself (i.e., the constraints capture those major things that must be true of the specialist's design before it can be considered to be successfully completed). Other constraints, embedded in the specialists plans, are used to check the correctness of intermediate design decisions. Still more constraints are present in the design data-base as general consistency checks. A typical specialist is shown in Figure 10.

The Selectors will be used to select from amongst the specialists plans. If no selector is specified a

```
(SPECIALIST
    (NAME       Head)
    (USED-BY AirCylinder)
    (USES       None)
    (DESIGN-PLANS        HeadDP1)
    (DESIGN-PLAN-SELECTOR  Headdpselector)
    (ROUGH-DESIGN-PLANS    HeadRDP1)
    (INITIAL-CONSTRAINTS   None)
    (FINAL-CONSTRAINTS     None)
```

Figure 10: *Specialist "Head". The code specifies the location of the design specialist in the hierarchy, names the design plans that are used by it and the constraints that its parameters may need to satisfy.*

default selector will be used which selects plans in declaration order. Note that in this declaration, as with others, the order of the individual parts of the declaration may vary according to the user's wishes.

Plans

Each specialist has a collection of plans that may be selected depending on the situation, and it will follow the plan in order to achieve that part of the design for which it is responsible. A Plan consists of a sequence of calls to Specialists or Tasks (see below), possibly with interspersed constraints. It represents one method for designing the section of the component represented by the specialist. The specialists below will refine the design independently, tasks produce further values themselves, constraints will check on the integrity of the decisions made, while the whole plan gives the specific sequence in which the agents may be invoked. Typically as one goes down in the hierarchy, the plans tend to become fewer in number and more straightforward. An example of this is shown Figure 11.

```
(PLAN
    (NAME    HEADDP1)
    (TYPE    Design)
    (USED-BY  Head)
    (SPONSOR  HeadDP1Sponsor)
    (BODY HeadTubeSeat)
         MountingHoles
         Bearings
         SealAndWiper
         AirCavity
         AirInlet
           (CHECK-CONSTRAINT Air)
         TieRodHoles
(REPORT-ON Head)
```

Figure 11: *Plan "HeadDP1". The design plan specifies the specialist that uses it, what criteria should be met for it to be chosen, and the plan steps.*

The type of a PLAN can be "Design" or "RoughDesign" depending on which phase of the design the knowledge applies to. The SPONSOR's job is to give an opinion to a selector about how suitable this plan is given the current state of the design. The BODY contains the details of the plan, and consists of an ordered list of plan items. In this example the plan consists entirely of tasks, with the exception of a constraint test and the last item which is a function provided by DSPL to print out the attributes and values of some part of the design.

Steps, Tasks, and Constraints

We consider a Step to be a design agent that can make one design decision given the current state of the design and taking into account any constraints. For example, one step would decide on the material for some subcomponent, while another would decide on its thickness.

```
(STEP
     (NAME          AirCavityID)
     (USED-BY        AirCavity)
     (ATTRIBUTE-NAME HeadAirCavityID)
     (REDESIGNER    AirCavityIDRedesigner)
     (FAILURE-SUGGESTIONS
          (SUGGEST  (DECREASE RodDiameter))
          (SUGGEST  (DECREASE HeadBearingThickness))
          (SUGGEST  (CHANGE HeadMaterial
                    TO DECREASE MinThickness))

)
(COMMENT "Find air cavity internal diam")

(BODY
     (KNOWN
      BearingThickness
          (KB-FETCH     'Head  'HeadBearingThickness)
      RodDiameter
          (KB-FETCH     'Rod   'RodDiameter)
     HeadMaterial
          (KB-FETCH     'Head  'HeadMaterial)
     MiniThickness
          (KB-FETCH     HeadMaterial 'MinThickness)
)
{Continued in next figure}
```

Figure 12: *Step "AirCavityID".*

A typical STEP in AIR-CYL is given in Figures 12 and 13. The STEP is broken down into two figures for convenience in display, and should be treated as one figure. A brief explanation of the role of the knowledge in the STEP is as follows. The sample STEP is USED-BY the AirCavity task. The ATTRIBUTE-NAME is the attribute for which this step is to design a value; that is, the Internal Diameter of the Air Cavity in the Head of the Air Cylinder. If a failure occurs the REDESIGNER will attempt to recover from it by altering the value just selected for this step's attribute. The declaration REDESIGN NOT-POSSIBLE is also allowed. If the step itself fails the FAILURE-SUGGESTIONS get passed up to the controlling task in a failure message. The suggestions refer to attributes that might be the cause of the failure. Each item in the suggestion list is evaluated at failure time. This allows conditional suggestions such as (IF (. x y) THEN (SUGGEST ...)). However, if the suggestion includes an expression, as in (DECREASE xyz BY (+ pqr o.56)), then the SUGGEST function will arrange for the value to be computed. The actions DECREASE, INCREASE or CHANGE refer to attributes, such as RodDiameter. In the current system all attribute names must be unique.

The BODY of the step is divided in KNOWN and DECISION sections. The keywords KNOWNS and DECISIONS will work just as well, and, in general, singular or plural keywords may be used as

```
{Continued from previous figure}
(DECISIONS
 MaxRodRadius (VALUE+ (HALF RodDiameter))
 MaxBearingThickness
          (VALUE+ BearingThickness)
 AirCavityRadius
          (+ MinThickness
               (+ MaxRodRadius
                    MaxBearingThickness))
AirCavityID   (DOUBLE AirCavityRadius)
REPLY (TEST-CONSTRAINT ACID)
REPLY (KB-STORE
     'Head 'HeadAirCavityID AirCavityID)
)
))
```

Figure 13: *Step "AirCavityID" continued.*

required. The KNOWN section obtains the values from the design data-base by doing KB-FETCH. The KB-FETCH uses the component and attribute names. The single quote (i.e., ') is used to indicate that the name givən is to be used directly without evaluation, as opposed to the use of a variable (e.g., HeadMaterial) that should be evaluated prior to use (e.g., HeadMaterial) that should be evaluated prior to use (e.g., giving the value 'Aluminum").

The DECISIONS sections contains the design knowledge. It consists of variable-action pairs, where the action is evaluated and its value assigned to the variable. That variable may then be used in subsequent actions in the step. The variables set in the KNOWNS section may also be used. Arithmetic expressions use prefix operators. The function VALUE+ returns the value plus the positive tolerance of the value, and consequently provides the largest magnitude for that value. There are many other functions available.

There are two distinguished variable names. One is REPLY, the other COMMENT. A comment will act as a dummy assignment and will expect a string as the action. This is just a way of inserting a comment into the body of the step. A REPLY variable is used when there is no value produced by the action but a message showing success or failure is produced instead. The TEST-CONSTRAINT and the KB-STORE are two examples. The value calculated by the step is put into the design data-base with a KB-STORE. It can produce a failure message if a constraint in the design data-base fails. Any failure will stop the execution of the body and will cause the DECISION section to fail.

A task is a design agent which is expressed as a sequence of steps, possibly with interspersed constraints. It is responsible for handling the design of one logically, structurally, or functionally coherent section of the component; for example a seat for a seal, or a hole for a bolt.

A Constraint is an agent that will test for a particular relationship between two or more attributes at some particular stage of the design. Constraints can occur at almost any place in the hierarchy. For example, a constraint might check that a hole for a bolt is not too small to be machinable given the material being used

4.c.6 Other Features

The main purpose of this exposition is not to give a complete description of DSPL, but to give a feel for the task-specific nature of the tool. A few other essential features of the language will now be briefly described.

Failure Handling and Redesign capability is an important requirement for anything other than relatively simple design problems. DSPL does not have failure analysis capabilities, but it can accept explicit knowledge about how to handle different kinds of failures during design. All design agents detect their own failure, are able to determine what went wrong (at least superficially), attempt to see if they can fix it locally, do so if they can, and report failure only if all attempts fail. Agents which have some control over other agents can use those agents in their attempt to correct the detected problem.

Each kind of agent can have different kinds of reasons for failing. For example, a step finds that a decision violates some constraint, a task discovers that a step's failure can't be mended locally, a plan can fail if it is discovered that it's not applicable to the situation to which it is being applied, while a specialist can fail if all of its plans fail.

For every kind of failure a message giving details is generated and passed back to the calling agent. The message includes, wherever possible, suggestions about what might be done to alleviate the problem. As there are usually many kinds of problems that can occur, an agent will first look at the message to decide what went on below. This is done by the Failure Handler associated with the agent. Much of the failure analysis is provided by the system, but for some cases, for example for constraint failures, the user (that is the person using the plan language to write a design system) has to supply some details. For some conditions immediate failure can be specified, for others an attempt to redesign might be made.

Knowledge about how to recover from failure can be coded as a *redesigner*. There appears to be a difference between the 'most reasonable choice' knowledge encoded in the step and the 'most reasonable adjustment' knowledge encoded in the redesigner. The language provides a number of constructs for representing failure handling knowledge of the above types.

Sponsors and selectors: A *sponsor* is associated with a plan and it is responsible for estimating the suitability of a plan for a particular design situation. A selector takes the output of the sponsor and will decide whether or not to use the plan if it has been recommended as suitable. The sponsor is expected to provide a suitability value for the plan, and if it cannot, a "use of plan language" failure will occur.

The *selector* takes as input the names of the plans being considered and their suitabilities for use in the design situation as decided by their sponsors. It will pick a plan for the specialist to execute.

4.c.7 Use and Extensions of DSPL

This section has discussed the idea of languages in which to express problem-solving knowledge, and has presented the language DSPL for a class of design problem-solving. DSPL embodies an underlying theory of routine design problem-solving. Examples of Specialist, Plan, Task, Step, Constraint, Redesigner, Failure Handler, Sponsor, and Selector knowledge were given. Most of these examples were taken from AIR-CYL, an expert system to design an Air Cylinder.

DSPL has been used for the construction of MPA, a system for routine logistics planning [11], and in the construction of a design system in the domain of chemical engineering [32].

More work needs to be done to test the applicability of this language to other design problems and other domains. Extensions to the theory must be made in order to handle design activity which is not of the type where both problem-solving and knowledge are known in advance. We feel that the identification of some of the types of design knowledge and their use is a substantial contribution towards understanding routine design activity.

5 The Generic Task Toolset

5.a Important Properties of the Toolset

So far, we have outlined the generic task theory and also described two such generic task tools: CSRL and DSPL. Tools corresponding to other generic tasks are also in existence in varying degrees of completeness. These tools are currently available for the Interlisp/Loops environment in the Xerox 1100 series of Lisp machines. CSRL and DSPL are also available in versions that are compatible with the KEE development system of Intellicorp. CSRL is also available in Commonlisp[7]. Currently, a project is under way at our Laboratory for the entire toolset to be made available in Commonlisp.

The integrated generic task toolset is extensible in the sense that more generic tools can be added as they are invented and additional problem solvers can be invoked as needed. The tools are intended to ensure the following advantages of the generic tasks, as described in [8].

- *Multiformity.* The more traditional architectures for the construction of knowledge based systems emphasize the advantages of uniformity of representation and inference. However, in spite of the advantage of simplicity, we argued earlier that uniformity results in a level of abstraction problem. A uniform representation cannot capture important distinctions between different kinds of problems. A uniform inference engine does not provide different control structures for different kinds of problems.

  The generic task approach provides multiformity. Each generic task provides a different way to organize and use knowledge. The knowledge engineer can choose which generic task is the best for performing a particular function, or can use different generic tasks for performing the same function. Different problems can use different generic tasks and different combinations of generic tasks.

- *Modularity.* A knowledge-based system can be designed by making a functional decomposition of its intended problem solving into several cooperating generic tasks, as illustrated in our discussion on diagnosis. Each generic task provides a way to decompose a particular function into its conceptual parts, e.g., the categories for hierarchical classification, and allows domain knowledge of other forms to be inserted into a generic task, e.g., evidence combination knowledge in hierarchical classification [39]. Each generic task localizes the knowledge that is used to satisfy local goals.

- *Knowledge Acquisition.* Each generic task is associated with its own knowledge acquisition strategy for building an efficient problem solver [7]. For example in hierarchical classification, the knowledge engineer needs to find out what specific categories should be contained in the

[7]CSRL is available as a supported product from Battelle Memorial Laboratories, Artificial Intelligence Group, in Columbus, Ohio, including versions in Commonlisp.

classification hierarchy and what general categories provide the most leverage for the establish-refine strategy.

- *Explanation.* This approach directly helps in providing explanations of problem solving in expert systems in two important ways: how the data match local goals and how the control strategy operates [15]. Also, the control strategy of each generic task is specific enough for generating explanations of why the problem solver chose to evaluate or not to evaluate a piece of knowledge. This is because of the higher level of abstraction in which control is specified for generic tasks.

- *Exploiting Interaction between Knowledge and Inference.* Rather than trying to separate knowledge from its use, each generic task specifically integrates a particular way of representing knowledge with a particular way of using knowledge. This allows the attention of the knowledge engineer to be focused on representing and organizing knowledge for performing problem solving.

- *Tractability.* Under reasonable assumptions, each generic task generally provides tractable problem solving [1, 21]. (One major exception is abductive assembly, which can become intractable under certain conditions, making it hard then for humans and machines to perform the task.) The main reasons why they are tractable are that a problem can be decomposed into small, efficient units, and knowledge can be organized to take care of combinatorial interactions in advance.

It should be noted that these advantages are attained at the cost of generality. Each generic task is purposely constrained to perform a limited type of problem solving and requires the availability of appropriate domain knowledge.

5.b Integrating and Combining GT's in an Application

It is hard at this overview level to give enough details of how the tools are to be combined to put together complex applications. There are both significant theoretical issues of integration as well as practical issues of technology and implementation in this regard. The diagnosis example is a good example to discuss the practical issue of how the tools in the toolset can be combined for an application.

We need to make the following distinctions that will be helpful here. Each tool such as CSRL can be regarded a a "shell" of a particular p.s. type. When the tool is invoked and knowledge and inference encoded using the knowledge primitives and message types, we have a *problem solver*, and different problem solvers built with the same shell can (and will typically) exist in an application. Each of the problem solvers is a specialist in two different senses: it specializes in a particular body of knowledge *and* in a type of problem solving, e.g., the *automech* (domain/concept) hierarchical classifier (type of problem solving) built out of CSRL, or the *BadFuel* (domain/concept) hypothesis matcher (type of problem solving) built out of HYPER, or the *AIRCYL* (domain/concept) hierarchical designer (type of problem solving). A given problem solver will typically need to access other problem solvers for information to continue its problem solving, e.g., the *BadFuel* matcher will need to know if various specific data items are present and for this it will need to access the data inference problem solver built from the tool IDABLE.

When the KBS is being built using the tools in the toolset, the knowledge engineer controls and directs the interaction among problem solvers by shaping and directing the messages appropriately. Without getting into details, the idea is simple: Each problem solver is characterized by (i) the kind of questions it can accept: e.g., the *Badfuel matcher* can accept messages that concern confidence values about the *Badfuel* concept and (ii) the kind of questions that it can ask, e.g., *Badfuel* concept can ask the database problem solver for values of specific data attributes. In the current version of the toolset these decisions have to be explicitly made by the knowledge engineer, i.e., which problem solver has to send what types of queries to what other problem solvers has to be specified at the time the system is being built. The toolset itself is built on top of a substratum that is object- and message-oriented so that building additional generic tools within the framework is a straightforward thing to do. See [26] for details on how the toolset is built up in this way.

For a complex application which involves portions which match the problem solving behaviors of the tools in the toolset, but which also has portions which require other methods of reasoning and represention not included within the current toolset, escaping to the object level or even the Lisp level (as in current implementations) to program AI or numerical techniques may be necessary and the toolset implementation supports this.

6 Concluding Remarks

In the late 70's, when we embarked on this line of research -- characterized by an attempt to identify generic tasks and the forms knowledge and control required to perform them -- the dominant paradigms in knowledge-based systems were rule and frame type architectures. While our work on use-specific architectures was evolving, dissatisfaction at the limited vocabulary of tasks that these architectures were offering was growing at other research centers. Clancey [17] in particular noted the need for specifying the information processing involved by using a vocabulary of higher level tasks. Task-level architectures have been gathering momentum lately: McDermott and his coworkers [28] have built SALT, a shell for a class of design problems, where critiquing proposed designs by checking for constraint-violations is applicable. Clancey [18] has proposed a shell called Heracles which incorporates the heuristic clasification strategy for diagnosis. Bennett [2] presents COAST, a shell for the design of configuration problem solving systems. All these approaches share the basic thesis of our own work, viz., the need for task-specific analyses and architecture support for the task. However, there are some differences in assumptions and methodology in some cases that needs further discussion.

The following conceptual distinctions are useful:

- "Building blocks" out of which more complex problem solvers can be composed, such as the tasks in the theory presented in the paper.
- Explicit high level strategies which we want a system to follow, where the strategies are expressed in terms of some set of tasks. Clancey's Heuristic Classification is an example of this. McDermott's and Marcus' Salt system uses a strategy called "propose and refine" which is also of this type.
- Compound tasks, such as the form of diagnosis described in earlier in the paper. An architecture for this compound task will bring with it its constituent generic tasks and also help in integrating the problem solving from the viewpoint of the overall task. In our

Laboratory, we are at work on building such diagnostic-level problem solving architectures for diagnosis of process engineering systems.

- Tasks which do not necessarily correspond to those human experts do well, but nevertheless can be captured as appropriate combinations of knowledge and inference and a clear function can be associated with them, e.g., constraint satisfaction schemes. Bennet's Coast system is an example of this.

Once we identify task-level architectures as the issue for highest leverage, then a number of immediate questions arise: what is the criterion by which a task is deemed to be not only generic but is appropriate for modularization as an architecture? How about an architecture for the generic task of "investment decisions"? Diagnosis? Diagnosis of process control systems? Is uncertainty management a task for which it will be useful to have an architecture? Are we going to proliferate a chaos of architectures without any real hope of reuse? What are the possible relationship between these architectures? Which of these architectures can be built out of other architectures? I do not propose to answer all these questions here, but they seem to be the appropriate kinds of questions to ask when one moves away from the comfort of universal architectures and begins to work with different architectures for different problems.

At this stage in the development of these ideas, empirical investigation of different proposals from the viewpoint of usefulness, tractability and composability is the best strategy. From a practical viewpoint, any architecture that has a useful function and for which one can identify knowledge primitives and an inference method ought to be considered a valid candidate for experimentation. As the tools evolve, one may find that some of the architectures are further decomposable into equally useful, but more primitive, architectures; or that some of them do not represent particularly useful functionalities, and so on.

The generic tasks that are represented in our toolbox were specifically chosen to be useful as technology for building diagnosis, planning and design systems with compiled expertise. For capturing intelligent problem solving in general, we will undoubtedly require many more such elementary strategies and ways of integrating them. For example, the problem solving activities in qualitative reasoning and device understanding, e.g., qualitative simulation, consolidation, and functional representation All these tasks have well-defined information processing functions, specific knowledge representation primitives and inference methods. Thus candidates for generic information processing modules in our sense are indeed many.

What does all this mean for an architecture of intelligence?

I am led to a view of intelligence as an interacting collection of *functional units*, each of which solves an information processing problem by using knowledge in a certain form and corresponding inference methods that are appropriate. Each of these units defines an information processing faculty. I discuss elsewhere [14] the view that these functional units share a computational property: they provide the agent with the means of transforming essentially intractable problems into versions which can be solved efficiently by using knowledge and inference in certain forms. For example, Goel et al [21] show how classification problem solving solves applicable cases of diagnosis with low complexity, while diagnosis in general is of high complexity. Knowledge is indeed power, but how it acquires its power is a far subtler story than the first generation knowledge based systems made it appear.

This view generates its own research agenda: As a theory, the generic tasks idea has quite a bit of work ahead of it in terms of a coherent story about how the tasks come together, are integrated and how more complex tasks such as planning come about from more elementary ones. How complex inference methods develop from simpler ones and how learning shapes these functional modules are issues to be investigated.

The relationship of task-specific architectures such as the GT ideas in this paper to, on one hand, more general architectures, such as SOAR [33], and on the other to "weak methods" is an intriguing one. My view is that from the perspective of modeling cognitive behavior, a GT-level analysis provides two closely related ideas which give additional content to phenomena at the SOAR architecture level. On the one hand, the GT theory provides a vocabulary of goals that a SOAR-like system may have. On the other hand, this vocabulary of goals also provides a means of indexing and organizing knowledge in Long Term Memory such that when SOAR is pursuing a problem solving goal, appropriate chunks of knowledge and control behavior are placed in Short Term Memory for SOAR to behave like a GT problem solver. In this sense a SOAR-like architecture, based as it is on goal achievement and universal subgoaling, provides an attractive substratum on which to implement future GT systems. In turn, the SOAR-level architecture can give graceful behavior under conditions that do not match the highly compiled nature of GT-type problem solving.

Acknowledgements

I am indebted to my colleagues with whom I have written a number of papers over the years on the generic task approach. In the preparation of this paper I have used excerpts from such papers. In particular the papers by the author and W. Punch, the author and D. C. Brown, the author, T. C. Bylander and John Josephson have been used. I acknowledge the assistance of Richard Fox and Vibhu Mittal in the preparation of this version of the paper. Matt Dejongh helped with some sections of this paper. The comments of John Fox, the editor of *Knowledge Engineering Review*, have helped improve the paper substantially. I also gratefully acknowledge the support of the Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010, and the Air Force Office of Scientific Research, grant 87-0090.

References

[1] Dean Allemang, Michael C. Tanner, Tom Bylander, and John R. Josepshson.
On the Computational Complexity of Hypothesis Assembly.
January, 1987
Proceedings of IJCAI-87, Milan, Italy, August, 1987.

[2] James Bennett.
COAST: A Task-Specific Tool for Reasoning About Configurations.
In *Proc. AAAI Workshop on High-Level tools.* AAAI, Shawnee Park, Ohio, 1986.

[3] Brown, D. C.
Expert Systems for Design Problem-Solving using Design Refinement with Plan Selection and Redesign.
PhD thesis, The Ohio State University, August, 1984.

[4] Brown, David C. and Chandrasekaran, B.
Knowledge and Control for a Mechanical Design Expert System.
IEEE Computer 19:92-101, July, 1986.

[5] Buchanan, B., Sutherland, G. and Feigenbaum, E.A.
Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses.
Organic Chemistry , 1969.

[6] Bylander, T., and S. Mittal.
CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling.
AI Magazine 7(3):66-77, 1986.

[7] Bylander, T., Smith J. and Svirbely, J.
Qualitative Representation of Behavior in the Medical Domain.
In *Proceedings of The Fifth Conference on Medical Informatics*, pages 7-11. Conference on Medical Informatics, Washington, D.C., October 26-30, 1986.

[8] Tom Bylander and Todd R. Johnson.
Structured Matching.
1987
OSU CIS LAIR Technical Report.

[9] Chandrasekaran, B.
Towards a Taxonomy of Problem Solving Types.
AI Magazine :9-17, Winter/Spring, 1983.

[10] Chandrasekaran, B.
Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design.
IEEE Expert 1(3):23-30, 1986.

[11] Chandrasekaran, B., and Tanner, M.
Uncertainty Handling in Expert Systems: Uniform vs. Task-Specific Formalisms.
Uncertainty in Artificial Intelligence.
North Holland Publishing Company, 1986, pages 35-46.
Kanal, L.N. and Lemmer, J. (Editors).

[12] B. Chandrasekaran.
Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1183-1192. Milan, Italy, August, 1987.

[13] Chandrasekaran, B., Smith, J., Sticklen, J.
Deep Models and Their Relation to Diagnosis.
Technical Report, , 1987.
Invited paper, Toyota Foundation Symposium on Artificial Intelligence in Medicine, Tokyo, Japan, August 1986, available as technical report from Laboratory of AI Research, Ohio State University.

[14] Chandrasekaran, B.
What Kind of Information Processing Is Intelligence? A Perspective On AI Paradigms and A Proposal.
This paper will appear in Source Book on the Foundations of AI, Partridge and Wilks, Editors, Cambridge University Press.
1987

[15] Chandrasekaran, B., M. C. Tanner and J. R. Josephson.
Explanation: the Role of Control Strategies and Deep Models.
Expert Systems: The User Interface.
Ablex Publishing Corporation, Norwood, New Jersey 07648, 1988, pages 219-247.
Editor, James Hendler.

[16] Chandrasekaran, B.
Design: An Information Processing Level Analysis.
Technical Report, The Ohio State University, Columbus, Ohio 43210, 1988.

[17] Clancey, W.J.
NEOMYCIN: Reconfiguring a Rule-Based Expert System for Application to Teaching.
In *Proc. Seventh International Joint Conference on Artificial Intelligence*, pages 829-836. IJCAI, Vancouver, 1981.

[18] Clancey, W. J.
Heuristic Classification.
Artificial Intelligence 27(3):289-350, 1985.

[19] Duda, R.O., Gaschnig, J.G., and Hart, P.E.
Model Design in the Prospector Consultant System for Mineral Exploration.
Expert System in the Microelectronic Age.
Edinburgh University Press, 1980, pages 153-167.
Editor, Michie, D.

[20] Friedland, P.
Knowledge-based Experiment Design in Molecular Genetics.
PhD thesis, Stanford University, Computer Science Department, 1979.

[21] Goel, A., N. Soundararajan, and B. Chandrasekaran.
Complexity in Classificatory Reasoning.
In *Proc. National Conference on Artificial Intelligence*, pages 421-425. Seattle, Washington, July 13-18, 1987.

[22] Gomez, F. and B. Chandrasekaran.
Knowledge Organization and Distribution for Medical Diagnosis.
IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1):34-42, January, 1981.

[23] Hashemi, S., Hajek, B.K., Miller, D.W., Chandrasekaran, B, and Josephson, J.R.
Expert Systems Application to Plant Diagnosis and Sensor Data Validation.
In *Proceedings of the Sixth Power Plant Dynamics Control and Testing Symposium.* Knoxville, Tennessee, April, 1986.

[24] Johnson, T.
HYPER: The Hypothesis Matcher Tool.
In *Proceedings of Expert Systems Workshop*, pages 122-126. Defense Advanced Research Projects Agency, Pacific Grove, CA, April 16-18, 1986.

[25] Johnson, K., Sticklen, J., and Smith, J.W.
IDABLE -- Application of an Intelligent Data Base to Medical Systems.
In *Proceedings of the AAAI Spring Artificial Intelligence in Medicine Symposium*, pages 43-44. American Association for Artificial Intelligence, Stanford University, March 22-24, 1988.

[26] Josephson, J.R., Smetters, D., Welch, A.K., Fox, R., Flores, G, and Lyndes, D.
Generic Task Toolset DRACO Release - Beta Test Including RA.
Technical Report, The Ohio State University, Computer & Information Science Department, Laboratory for Artificial Intelligence Research, February, 1988.

[27] Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C.
A Mechanism for Forming Composite Explanatory Hypotheses.
IEEE Trans. on Systems, Man and Cybernetics :pp.445-454, 1987.

[28] Marcus, Sandra, and John McDermott.
SALT:A Knowledge Acquisition Tool for Propose-and-Revise Systems.
Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1987.

[29] J. McDermott.
R1: A Rule-based Configurer of Computer Systems.
Artificial Intelligence 19(1):39-88, 1982.

[30] Miller, R.A., Pople, H.E., and Meyers, J.D.
Internist-1, An Experimental Computer-based Diagnostic Consultant for General Internal Medicine.
Readings in Medical Artifical Intelligence.
Addison-Wesley Publishing, 1984, pages 190-209.

[31] Mittal, S.
Design of A Distributed Medical Diagnosis and Data Base System.
PhD thesis, The Ohio State University, 1980.

[32] Myers, D.R., Davis, J.F., and Herman, D.
A Task Oriented Approach to Knowledge-Based Systems for Process Engineering Design.
Computers and Chemical Engineering, Special Issue on AI in Chemical Engineering Research and Development : , August, 1988.

[33] Laird, J.E., Newell, A., Rosenbloom, P.S.
SOAR: An Architecture for General Intelligence.
Artificial Intelligence 33:1-64, 1987.

[34] Punch, W. F., Tanner, M. C., Josephson, J. J.
Design Considerations for PEIRCE, A High-Level Language for Hypothesis Assembly.
Expert Systems in Government Symposium :279-281, October, 1986.

[35] Shortliffe, E.H.
Computer-based Medical Consultations: MYCIN.
Elsevier/North-Holland Inc., 1976.

[36] Shum, S.K., Davis, J.F., Punch III, W.F., and Chandrasekaran, B.
An Expert System Approach for Malfunction Diagnosis in Chemical Plants.
Computers and Chemical Engineering 12(1):27-36, 1988.

[37] J. W. Smith, M.D., J. R. Svirbely, C. A. Evans, P. Straum, J. R. Josephson and M. C. Tanner.
RED: A Red-Cell Antibody Identification Expert Module.
Journal of Medical Systems 9(3):121-138, 1985.

[38] Sticklen, J., Chandrasekaran, B. and Josephson, J.
Control Issues in Classificatory Diagnosis.
In *Proceedings of The 9th International Joint Conference on Artificial Intelligence.* International Joint Conference on Artificial Intelligence, University of California, Los Angeles, CA, August 18-24, 1985.

[39] Jon Sticklen.
MDX2: An Integrated Medical Diagnostic System.
June, 1987
Phd. Dissertation, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210.

[40] Tanner, M. and Bylander, T.
Application of the CSRL Language to the Design of Expert Diagnosis Systems: The Auto-Mech Experience.
Artificial Intelligence in Maintenance.
Noyes Publications, Park Ridge, N.J., 1985.

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
August 1987/Revised January 1988

Design: An Information Processing-Level Analysis

B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

Note: This is a draft version of Chapter 2 of the book *Design Problem Solving: Knowledge Structures and Control Strategies* by D. C. Brown and B. Chandrasekaran (forthcoming).

ABSTRACT

Design problem solving is analyzed as an information-processing task: the task and its information requirements are analyzed. This analysis suggests possible decompositions of the task into a number of subtasks, depending upon what kinds of knowledge are in fact available in a domain. This decomposition can be carried on several levels until we reach an understanding of how various generic problem solving capabilities can come together to help solve the design problem. This analysis suggests possible problem solving architectures for design. A number of AI approaches to design are discussed in this perspective and it is shown how each of then can be understood as solving a particular version of the design problem, using one of the architectures that arises from the analysis in such a way that the architecture matches the knowledge available in the domain.

Table of Contents

2. Design: An Information Processing-Level Analysis

2.1. A Framework for Design Problem-Solving

2.1.1. What is the Design Problem?

In this chapter we look at design as an information processing task: i.e., specify what kinds of input and output characterize the task abstractly. This can then form the basis for investigating what kinds of knowledge and inference processes can help solve what parts of the task. We will avoid talking in terms of particular methods of representation of knowledge, say, rules or frames, but concentrate instead on *what* needs to be represented, and what types of inferences are needed. Once the nature of the subtasks in design becomes clear, then the question of how to implement them can be undertaken. The *reader might remember that one of our criticisms of the expert system area has been that implementation* level phenomena have been allowed to interfere with an analysis of task-level phenomena. We would like to keep them clearly separate.

Design is a very complex activity and covers a wide variety of phenomena: from planning a day's errands to theory construction in science to composing a fugue are design activities. In order to give some focus and use some shared intuitions, let us restrict the scope in this discussion to the design of artifacts that satisfy some goals.

A designer is charged with specifying how to make an artifact which satisfies or delivers some goals. For each design task, the availability of a (possibly large and generally only implicitly specified) set of *primitive components* can be assumed. The domain also specifies a repertoire of *primitive relations* or connections possible between components. An electronics engineer, e.g., may assume the availability of transistors, capacitors, etc., of various types when he is designing a waveform generator, and examples of primitive relations in that domain are serial and parallel connections between components. Similarly, an architect might assume the availability of building materials. If the architect has to design an unusual brick as part of his architectural specification, at least he can assume the availability of clay, and so on. Of course design can also be recursive: if a certain component that was assumed to be available is in fact not available, the design of that can be undertaken at the next round, and the domain for the component may be rather different than the original domain, as in the example of building and brick design. If the component design is not successful, the original design may be discarded and the task undertaken again.

The design task can then be specified as follows:

- *Complete specification of a set of 'primitive' components and their relations so as to meet a set of constraints.*

Some of the constraints will refer to the functions or goals of the artifact, some may pertain to the parameters of the artifact (e.g., 'total weight to be less than 1 ton'), yet others may provide constraints on the design process itself, and finally constraints may apply to the process of assembling the artifact (manufacturing constraints). Often the goals may not be stated explicitly or in sufficient detail at the start of the design process. In hard design problems, the world of primitive objects may be very open-ended. *In spite all such caveats, the above working definition* is a good starting point for our discussion.

This definition also captures the *domain-independent* character of design as a generic activity at some level of abstraction. Planning, programming and mechanical design all share the above definition to a significant degree. Note that the knowledge needed and many of the detailed mechanisms will of course be domain-specific. For example, mechanical design may call for significant amounts of spatial reasoning, while the electrical domain may only involve topological reasoning. But the nature of the design problem as a whole has many commonalities at the level of the above definition, and as we shall see, at the level of many of the subprocesses.

This definition is not meant to imply the existence of one method for all design. The main message of our work is that design actually consists of a large number of distinct processes that work together, each contributing some information needed during the design process. In fact the apparent difference in the design process in different domains and different designers can be explained by the dominance of some of these subprocesses over the others due to differences in the knowledge available.

The above definition gives a clue to why the design problem can be *hard* for AI and often also for people. In realistic domains the size of the set of primitive objects is quite large and implicit. The design problem is formally a search problem in a very large space for objects that satisfy multiple constraints. Only a vanishingly small number of objects in this space constitute even "satisficing", not to speak of optimal, solutions. What is needed to make design practical are powerful strategies that radically shrink the search space.

2.2. What Kind of Space in Which To Search?

The idea of search in a state space goes back to the early days of AI, and Newell [Newell, 1980] has formalized the *Problem Space Hypothesis* essentially stating that all goal-directed behavior takes place in some problem space.

Before search can take place, the problem space needs to be defined. But design problem solving does not have a unique problem space. Different kinds of problem spaces can be visualized, each appropriate for some kinds of domain knowledge and not others. For search in a problem space to be operationally definable, problem states, operators which transform one problem state into a set of successors, and some ordering knowledge that helps to choose between alternatives need to be available. For search to be practical, generation of successors and choice among alternatives should not themselves be complex problem solving activities. The last condition means that domain knowledge should be directly available which can be applied to generate successors and choose among them.

Let us consider the *transformation* approach to design [Barstow, 1984;Balzer, 1981] as a concrete illustration of these issues. We can consider the set of specifications to be the initial state, and a fully designed artifact to be the end state. Operators transform parts of the specifications into alternative design commitments that will realize them. So an intermediate state will consist of design commitments which realize some of the specifications along with remaining specifications. The process of design can be thought of as searching for a series of design commitments that results in a goal state.

While this is formally satisfactory, knowledge may not be available in all domains for successor generation and alternative selection. In the programming domain to which this idea has been applied, there seem to be several examples where knowledge of this form is in fact available. However, this problem space is not of general applicability. (No single problem space is.) In some domains, the constraints as stated may not be factorizable in this way, and there may be significant interactions between the designs that are chosen to meet parts of the constraints. There is also no guarantee that the design process can always correspond to incremental choices. Large subsystems may be designed first and then only design within subsystems may proceed. Thus the actual design process in that domain may not correspond to navigation in this transformation problem space. Knowledge may be directly available which cuts a swath across the space, so that several constraints together are realized by a precompiled design that is recognized as applicable (design plans). Finally, in many domains, the problem is reformulated by a decomposition so that a number of disjoint local spaces, each corresponding to a subproblem are created. (We will discuss decomposition and design plans shortly in greater detail.)

The point of the discussion is this. Which problem space is used depends on the forms in which domain knowledge for representation and control are available. Using an inappropriate problem space will result in artificial heuristic functions being used which do not capture the real structure of domain knowledge.

We propose that in design problem solving *a variety of types of knowledge* can be identified, each of wnich helps solve a portion of the design problem in a computationally efficient way. Expertise consists of an accumulation of a repertoire of such knowledge. However, unlike the current view in the expert systems area, this expertise is not viewed as collections of pieces of knowledge, to be used by a uniform inference technique. Instead, knowledge comes in various *generic forms*, each structured in characteristic ways and using inference methods that are appropriate to it. Each type of knowledge can produce some information that may be needed or useful during design, or can generate a part of the design solution. Conversely, each type of knowledge *requires* information of certain types to be available before it can be useful.

Thus the picture that we would like to give of design problem solving is as a cooperative activity between multiple types of problem solvers, each solving a subproblem using knowledge and inference of specific types, and communicating with other computational modules or problem solvers for information *that is needed for it to perform its task, or to deliver information that they need for their* tasks. Thus an analysis of design as problem solving consists of identifying these subprocesses, their information processing responsibilities, and the knowledge and inference needed to deliver these functions. We call this kind of analysis an *information processing analysis* of design. This is the task of the next section.

2.3. Information processing Analysis of Design

The style of analysis will be to identify subtasks in design, and characterize what kinds of information or solution they are responsible for providing. Some of these subtasks can be performed in a number of different ways: an AI solution is only one way. For example, during design, it will be necessary to find if a certain design requirement is met. A traditional computational algorithm may be able to do that in some cases, e.g., finding out if stress in a member is less than a certain amount may be done by invoking a finite element analysis algorithm. Sometimes this information may require an AI-type solution, involving an exploration of some space in a qualitative way, e.g., by doing a qualitative simulation of the artifact. In what follows we will only describe issues associated with AI-type solutions for these subtasks, but the larger possibility needs to be kept in mind in the actual design of knowledge-based systems for design.

During the discussion we will try to relate the framework to a number of previous and current approaches to design. But the literature on design is vast. Even within AI, work on design has proliferated over the last decade. We do not intend to be exhaustive in our coverage. Our intent is to point to some of the other work as a way to illuminate the discussion.

We will describe a number of subprocesses or subtasks in design and describe the role they play in design. The design process can be usefully separated into those processes that play a role in the "generate" part and those that help in the "test" part. We subdivide our discussion into two groups of processes: those that are responsible for proposing or making design commitments of some sort, and those that serve an "auxiliary" role, i.e., generate information needed for the proposers, and help test the proposed design.

2.3.1. Processes That Propose Design Choices

(1): **Decomposition.** This a very common subpart of the design activity. We will use this process as an example of information processing analysis, and describe it in terms of all the features that such an analysis calls for: types of knowledge, information needed, and the inference processes that operate on this form of knowledge.

Knowledge of the form D --> {D1, D2, .. Dn}, where D is a given design problem, and Di's are "smaller" subproblems (i.e., associated with smaller search spaces than D) is often available in many domains. In many domains, there may be a number of alternate decompositions available, and so choices (and possible backtracking) will need to be made in the space of possible decompositions. Repeated applications of the decomposition knowledge produce *design hierarchies.* In well-trodden domains, effective decompositions are known and little search at that level needs to be conducted as part of routine design activity. For example, in automobile design, the overall decomposition has remained largely invariant over several decades.

Dependable decomposition knowledge is extremely effective in controlling search since the total search space is now significantly smaller. This power arises from the fact that decompositions represent a previously compiled solution to a part of the design problem, and thus at run-time the design problem solver can avoid this search.

Information Needed: The decomposition process needs two kinds of additional information for it to be effective.

- How the goals or constraints on D get translated into constraints on the subproblems D1,... Dn.

- How to glue the designs for D1, D2,.. Dn into a design for D.

Information of the above types may be given as part of the decomposition knowledge or can be obtained by accessing another processor which can produce that information. We will shortly refer to a method called *constraint posting* that has been proposed for generating constraints on subproblems. How to glue the designs for subproblems may require additional problem solving, such as simulating D1 and D2, e.g., and finding out exactly where and how the gluing can occur. The Critter system [Kelly, 1984], e.g., provides such a simulation facility applicable under certain assumptions that helps both in generating constraints for the subproblems and in gluing the solutions together.

Inference Process: There are two sets of inference processes, one dealing with which sets of decompositions to choose, and the other concerned with the order in which the subproblems within a given decomposition ought to be attacked. (Remember that a decomposition merely converts a design problem into a set of presumably "smaller" problems, which still need to be solved for the decomposition to be successful.)

For the first problem, in the general case, the decomposition will produce an AND/OR node, i.e., will produce decompositions some of which are alternatives and others all of which need to be solved. Finding the appropriate decomposition may involve searching in a space represented as an AND/OR graph. But as a rule such searches are expensive. Routine design problems should not require extensive searches in the decomposition space. To avoid the search problem but to use domain knowledge about decomposition, human-machine interaction between human experts and machine processing can be arranged so that the machine proposed alternative decompositions, and the human chooses the most plausible ones. Precisely this sort of shared labor is used in the VEXED system [Mitchell, etal, 1985] during its problem decomposition phase.

The problem of the order in which to attack the problems in the decomposition list when combined with the problem of searching in the decomposition space can make the total search very complex, since the investigation of the subproblems in a given decomposition will be in general non-reusable if that decomposition turns out not to be successful. This explains the extreme difficulty of the design problem in the general case. However, in most cases of routine design, the decomposition knowledge leads to a design hierarchy as mentioned. The default control process for investigating within a given design hierarchy is then *top-down.* While the control is top down, the actual sequence in which design problems are solved may occur in any combination of top-down and bottom up manner. For example, in designing an electronic device, a component at the tip level of the design hierarchy may be the most limiting component and many other components and subsystems can only be designed after that is chosen. The actual design process in this case will appear to have a strong bottom up flavor. Control first shifts to the bottom-level component, and the constraints that this component design places on the design of other components are passed up.

A related issue is one of whether the control should be depth-first or breadth first. Again, this is very much a function of the domain. The specification language for control behavior in this process should be expressive enough for a variety of control possibilities along these lines.

Decomposition is an ubiquitous strategy in AI work in design. McDermott's NASL system [D.V.McDermott, 1978] uses this extensively. Freeman and Newell [FreemanandNewell, 1971] discuss various decomposition criteria, including functional and structural. The transformational design work of Barstow and Mostow uses decomposition in a degenerate form: the constraint set is such that subsets of it correspond to different design problems, and so can be separately expanded.

(2): **Design Plans.** Another pervasive form of design knowledge, again that represents precompiled partial design solutions, is a *design plan.* A design plan specifies a sequence of design actions to take for producing a piece of abstract or concrete design. In abstract design, choices are made which need to be further "expanded" into concrete details at the level of primitive objects. These design plans are indexed in a number of ways, two of them being indexing by design goals (*for achieving <goal>, use <plan>*), or by components (*for designing <part>, use <plan>*). Since plans may have steps that point to other plans, design plans can subsume decomposition knowledge. From the viewpoint of complexity reduction, the central contribution plans make is as an encoding of previous successful exploration of a problem space by abstracting from the experience of an individual expert or a design community in solving particular design problems.

Each goal or a component may have a small number of alternate plans attached to them, with perhaps some additional knowledge that helps in choosing among them. A number of control issues arise about abandoning a plan and backing up appropriately, or *modifying* a plan when a failure is encountered.

The inference process that is applicable can be characterized as *instantiate and expand*. That is, the plan's steps specify some of the design parameters, and also specify calls to other design plans. Choosing an abstract plan and making commitments that are specific to the problem at hand is the instantiation process, and calling other plans for specifying details to portions is the expansion part.

A number of additional pieces of information may be needed or generated as this expansion process is undertaken. Information about dependencies between parts of the plan may need to be produced at runtime (e.g., discovering that certain parameters of a piston would need to be chosen before that of the rod), and some optimizations may be discovered at run time (e.g., the same base that was used to attach component A can also be used to attach component B). For example, Noah [Sacerdoti, 1977] can be understood as a system that instantiates and expands design plans. In Noah, corresponding to each goal of the artifact under design, there is a stored procedure which can be interpreted as a design plan. These plans can call other procedures/plans until a hierarchy is procedures is created. Noah concentrates its problem solving on recognizing ordering relations and redundancies between the components of the plan.

The idea of design plans has been used successfully in the domain of programming or algorithm design [Rich, etal, 1979], [JohnsonandSoloway]. The notion that *plans* constitute a very basic knowledge structure has been with us from the 1950's when this idea was discussed extensively by Miller, Galanter and Pribram [Miller, etal, 1960]. Schank and Abelson [SchankandAbelson, 1977] have also discussed the use of plans as a basic unit of knowledge. The Molgen work of Friedland [Friedland, 1979] uses design plans as a basic construct. More recently Mittal's PRIDE system [Mittal, etal, 1986] has used them for design knowledge representation.

(3): **Design by Critiquing and Modifying Almost Correct Designs.** A variation on the design plan idea is that the designer has a storehouse of actual successful designs indexed by the goals and constraints that they were designed to satisfy. Sussman [Sussman, 1973] has proposed that a design strategy is to choose an already-completed design that satisfies constraints closest to the ones that apply to the current problem, and modify this design for the current constraints. This process needs information of the following kinds.

- *Matching*: How to choose the design that is "closest" to the current problem? Some notion of prioritizing over goals or differences in the sense of *means-ends analysis* may be needed, if this information cannot be generated by a compiled matching structure. In some cases, some analogical reasoning capabilities may be appropriate by which to recognize "similar" problems.

- *Critiquing*: Why does the retrieved design fail to be a solution to the current problem? This analysis is at the heart of learning from failure, and sophisticated problem solving may be needed to analyze the failure. This capability of critiquing a design is of more general applicability than for this particular design process.

- *Modifying*: How to modify the design so as to meet with the current goals? In many cases, this information may be available in a compile form, but in general, this also requires sophisticated problem solving.

The processes of critiquing and modifying have more general applicability than as parts of this particular design process. We discuss criticism as one of the auxiliary processes later in this section. Design modification, however, is a useful process in the "generate" part of design, so we discuss some of the issues related to it here.

Modification as a subprocess takes as input information about failure of a candidate design and changes the design. Depending upon the sophistication about failure analysis and other forms of knowledge available, a number of problem solving processes are applicable:

- A form of means-ends reasoning, where the differences are "reduced" in order of most to least significant.
- A kind of hill-climbing method of design modification, where parameters are changed, direction of improvement noted, and additional changes are made in the direction of maximal increment in some measure of overall performance. This form can even constitute the only method of design in some domains: assign arbitrary values to the parameters, and change them in a hill-climbing fashion until a maximum is reached, and deliver that as the design. This is especially applicable where the design problem is viewed as a parameter choice problem for a predetermined structure. The system called DOMINIC [Howe, etal] engages in this form of design problem solving.
- Dependencies can be explicitly kept track of, in such a way that when a failure occurs, the dependency structure directly points to *where* a change ought to be made. *Dependency-directed backtracking* was proposed by Stallman and Sussman [1977] as one approach to this problem. Mittal [MittalandAraya, 1986] proposes a variation on dependency tracking for modification of designs on failure.
- What to do under different kinds of failures may be available as explicit domain knowledge in routine design problems. This information can be attached to the design plans. The work to be described in later chapters uses this highly compiled form.

Sussman [1973] has investigated the retrieval and use of previous designs in circuit design. Schank [1983] has been an advocate of case-based reasoning for a variety of problems.

(4): **Design by Constraint Processes.** For some design problems a process of simultaneous constraint satisfaction by constraint propagation can be employed. In order for this to work computationally effectively, it is best if the structure of the artifact is known and design consists of selecting parameters for the components. Constraints can be propagated in such a way that the component parameters are chosen to incrementally converge on a set that satisfies all the constraints. Macworth [1977] provides a good discussion of several techniques for this. This is an instance of what is called, in optimization theories, *relaxation procedures*[1]. Human problem solvers aren't particularly good at this form of information processing without pencil and paper. The incremental convergence process can be treated as a form of problem space exploration, so we are including it in this discussion.

Constraint satisfaction processes can be viewed as applying design modification repeatedly and incrementally. Thus many of comments we made earlier regarding design modification are applicable. In particular, some of the constraint propagation techniques can be viewed as versions of *hill-climbing* methods in search. And variations such as dependency-directed changes to parameters can be adopted during each modification cycle. More complex processes such as *constraint-posting* can be used where additional constraints are generated as a result of choices made for earlier parameter choices. These constraints are used for remaining parameter choices.

Config on problems are an interesting and well-known class of problems (made famous by the R1 system [J.McDermott, 1982]) in design. Some versions of them can be decomposed into subproblems

[1]Unfortunately, this use of the term relaxation interferes with another use of it in design, viz., *relaxing the constraints* so that a hard design problem may be converted into a relatively easier one.

whose solutions can be neatly glued back together. In fact, R1's problem solving is done as a linear series of subtasks. However, in the general case, these problems often have no clear decomposition into subproblems, because of extensive interactions between various parts of the design. On the other hand, many configuration problems have the tractable feature that most of the components of the device are already fixed, and only their connections and a few additional components to mediate the connections need to be chosen. This makes iterative techniques applicable by making it likely that one can converge on the solution. Constraint satisfaction methods are often applicable to configuration problems. Marcus, McDermott and Wang [1985] discuss a strategy called *propose and revise*, where commitments are made for some parts of the design, which generates additional constraints, and if later parts in the design problem cannot be solved, earlier commitments are revised. Frayman and Mittal [1987] discuss the configuration task abstractly.

Caution! Formally *all* design can be thought of as constraint satisfaction, and one might be tempted to propose global constraint satisfaction as a universal solution for design. The problem is that these methods still can constitute a fairly expensive way to search the space. For example, *propose and revise* can end up searching the entire space in difficult problem spaces and hill-climbing methods can get stuck at local optima. Hence these methods are not a universally applicable for practical design. Other methods of complexity reduction such as problem decomposition are still very important in the general case. They can create subproblems with sufficiently small problem spaces in which constraint satisfaction methods can work without excessive search.

Human problem solvers need computational assistance in executing constraint satisfaction approaches: the methods are computationally intensive and place quite a burden on short term memory. As long as attempts are not made to use them as universal design methods, they can be effective computational techniques for portions of the design problem.

2.3.2. Auxiliary Processes

So far the subprocesses in design that we have considered:

- decomposition, design plan instantiation and expansion, modification of an almost correct design, constraint satisfaction,

contribute to design by proposing some design commitments. Along the way, we have referred to some other processes which serve the former by providing information that they need. Let us discuss them briefly here.

(1): **Goal/Constraint Generation for Subproblems.** Given a decomposition D --> {D1, D2, .. Dn}, one will need to know how the goals/constraints of Dare translated into goals/constraints for the subproblems. In many domains, this information is part of the decomposition knowledge. But if it is not available, additional problem solving is called for. The literature on constraint-posting that we referred to earlier proposes methods applicable in some cases.

Vexed [Sternberg] provides an example of constraint generation for subproblems given a particular problem decomposition. In this domain the subproblems have a serial connection relation. For example, D may be implemented by two modules D1 and D2 connected in series. A constraint propagation scheme (called CRITTER [Kelly, 1984]) takes the input to D and produces the constraints on D1's output/ D2's input. Design of D1 and D2 can then proceed.

(2): **Recomposition.** We alluded to this in our discussion on decomposition: how to glue the solutions of the subproblems back into a solution for the original problem. Integrating them may require simulating the subdesigns and find how they interact. Or other methods of problem solving may be called for. Scientific theory building involves assembling portions of theories into larger coherent theories, and needs powerful interaction analyses. RED [Josephson, etal, 1987] proposes an especially powerful strategy for composing explanatory theories.

(3): **Design Verification:** This is part of the "test" component of the design activity: whether a candidate design delivers the functions and meets with any other relevant constraints. In most cases, it can be done by straightforward compiled computational methods, e.g., "add weights of components and check that it is less than x," or invoking possibly complex mathematical formulae, such as a finite element analysis, that does not involve problem solving. In some cases, additional problem solving may be called for for verification. For instance, qualitative simulation of a piece of machinery to decide if any of its parts will be in the path of another part may be needed for verifying a proposed design.

(4): **Design Criticism**. At any stage in design, any failure calls for analyzing the candidate design for reasons for failure. This form of criticism played a major role in the method of design by retrieving an almost correct design. In most routine design, fairly straightforward methods will suffice for criticism, but in general this calls for potentially complex problem solving. *Design modification* uses the results of criticism.

2.4. Implications of Above Analysis

The analysis of the design process in terms of subprocesses with well-defined information processing responsibilities helped us in identifying types of knowledge and inference needed. This in turn directly suggests a *functional architecture* for design with these subprocesses as *building blocks*. It also suggests a principled way in which to define the human-machine interaction in design. Firstly, whenever knowledge and control can be explicitly stated for one of the modules or building blocks, that module can be built directly, by using a knowledge and control representation that is appropriate to that task. Secondly, if knowledge for a module is not explicitly available, the human can be part of the loop for providing information that that module would have been responsible for. For example, failure analysis and common sense reasoning involving space and time are difficult problem solving tasks. These tasks may be needed for the performance of design modification and design verification, respectively. The human/machine division of responsibility may be done in such a way that the machine turns to the user for the performance of these tasks. As these tasks are better understood, they can be incrementally brought into the machine side of the human/machine division of labor.

Another kind of human machine interaction is possible is possible in this framework. Note that each subprocess is characterized both by specific types of knowledge and by inference and related control problems. We mentioned, e.g., that search in the space of problem decompositions can become quite complex. One way in which a module can interact with a domain expert is by proposing available knowledge and letting the human make the control choices by using knowledge that has not been made explicit in the problem solving theory. As a practical matter, this can be an effective way of using the module as a knowledge source, even without a complete theory of problem solving using that knowledge. The VEXED system that we have mentioned in fact works in this mode: it proposes possible decompositions, and the user is asked to choose the one he or she would like to pursue. Similarly, when a design system's choice of design plans fails, it may turn to the user for choosing alternative plans.

Let us elaborate on the functional architecture for design that results from this analysis. Because each subprocess uses characteristic types of knowledge and inference, a "mini-shell" can be associated with it and knowledge and inference can be directly encoded using that shell. Since each of the tasks has a clear information processing responsibility, the modules can communicate with each other in terms of the information that defines the input and outputs of these modules. Thus the modularity that results is a task-level modularity.

In the rest of this book, we provide the details of the functional architecture for one type of design, a form of *routine design* that we have termed Class 3 Design. This way of analyzing design and identifying architectures out which design problem solvers can be built is what is novel about the point of view of the book. We will soon proceed to a description of our informal classification of design problems, but before that we need to take note of some other suggestions that have been made for design problem solving and relate them to our analysis.

2.5. Classes of Design

The above analysis of design subprocesses can be used to provide an *informal* classification of design problems. Many of the processes in the "test" part of design, such as design verification by qualitative simulation, can be arbitrarily complex, but they are not particularly specific to design. The design process simply calls upon these other problem solving skills. On the other hand, many of the processes in the "generate" portion are quite specific to design as a problem solving process, so our classification is based largely on the subprocesses in the "generate" part of design.

Each of the processes

- decomposition, design plan instantiation and expansion, modification of similar designs, constraint satisfaction,

performs some aspect of design, using information either directly available or supplied by auxiliary problem solving or other computational processes. Each of them comes with a set of control problems that can be more or less complex, and needs knowledge in certain forms.

The framework suggests that design by decomposition, i.e., breaking problems into subproblems, by plan synthesis where necessary, and by plan selection where possible, are the core processes in knowledge-based design, i.e., it gives importance to the first two processes in the above list as the major engines of complexity reduction in design. The classification is largely based on the difficulty of these subtasks or processes, in particular on the completeness of knowledge, the ready availability of the needed auxiliary information and the difficulty of the control issues.

2.5.1. Class 1 Design

This is open-ended "creative" design. Goals are ill-specified, and there is no storehouse of effective decompositions, not to speak of design plans for subproblems. Even when decomposition knowledge is available, most of the effort is in searching for potentially useful problem decompositions. For each potential subproblem, further work has to be done in evaluating if a design plan can be constructed. Since the design problem is not routine, considerable problem solving for many of the auxiliary processes will need to be performed.

The average designer in industry will rarely, if ever, do Class 1 design, as we consider this to lead to a major invention or completely new products. It will often lead to the formation of a new company, division, or major marketing effort. This then is extremely innovative behavior, and we suspect that very little design activity is in this class.

2.5.2. Class 2 Design

Class 2 design is characterized by powerful problem decompositions already available, but design plans for some of the component problems in need of *de novo* construction or substantial modification. Design of a new automobile, e.g., does not involve new discoveries about decomposition: the structure of the automobile has been fixed for quite a long time. On the other hand, several of the components in it constantly undergo major technological changes, and routine methods of design for some of them may no longer be applicable.

Complexity of failure analysis will also take a problem away from routine design. Even if design plans are available, if the problem solver has to engage in very complex problem solving procedures in order to decide how to backtrack, the advantage of routine design is reduced. In short, whenever substantial modifications of design plans for components are called for, or when synthesis in the design plan space is especially complicated, we have a Class 2 problem.

2.5.3. Class 3 Design

This is relatively routine design: effective problem decompositions are known, compiled design plans for the component problems are known, and actions to take on failure of design solutions are also explicitly known. There is very little complex auxiliary problem solving needed. In spite of all this simplicity, the design task itself is not trivial: complex backtracking can still take place. The design task is still too complex for simple algorithmic solutions or table look up.

Class 3 problems are routine design problems, but still requiring knowledge-based problem solving. The ensuing chapters of this book deal with an approach to building knowledge-based systems for routine design problems of this type. The processes described here can work in conjunction with auxiliary problem solvers of various types, but the theory for them is not developed further in this book. The examples used all assume that the information to be provided by the auxiliary design processes, e.g., design criticism, verification, and subproblem constraint generation, are all available in a compiled manner.

2.5.4. A Class 3 Product

In a large number of industries, products are tailored to the installation site while retaining the same structure and general properties. For example, an Air-cylinder intended for accurate and reliable backward and forward movement of some component will need to be redesigned for every new customer in order to take into account the particular space into which it must fit or the intended operating temperatures and pressures. This is a design task, but a relatively unrewarding one, as the designer knows at each stage of the design what the options are and in which order to select them. Note that that doesn't mean that the designer knows the complete sequence of steps in time (i.e., the trace) in advance, as the designer has to be in the problem-solving situation before each decision can be made. There are just too many combinations of requirements and design situations to allow an algorithm to be written to do the job.

As this tends to be unrewarding work for humans and as this type of non-trivial problem appears to be possible to do by computer there is strong economic justification for us to attack this problem.

2.5.5. Class 3 Complexity

The complexity of the class 3 design task is due not only to the variety of combinations of requirements, but also to the numerous components and sub-components, each of which must be specified to satisfy the initial requirements, their immediate consequences, the consequences of other design decisions, and the constraints of various kinds that a component of this kind will have.

While class 3 design can be complex overall, at each stage the design alternatives are not as open-ended as they might be for class 2 or 1, thus requiring no planning during the design. In addition, all of the design goals and requirements are fully specified, subcomponents and functions already known, and knowledge sources already identified. For other classes of design this need not be the case. Consequently, class 3 design is an excellent place to start in an attempt to fully understand the complete spectrum of design activity.

Note that we are not merely interested in producing an expert system that produces a trace which is the same as or similar to a designer's, nor are we solely interested in arriving at the same design -- although both are amongst our goals. We are concerned with producing an expert system that embodies a theory of class 3 design and demonstrates the theory's viability.

Imprecision of the Classification: The classification that we have described is a useful way to get a bearing on the complexity of the design task, but it is *not* meant to be formal or rigorous. Neither is the term *routine design*. The approach described in this book is intended to provide a starting point for capturing some of the central phenomena in routine design, but it is not intended to be a complete account of routine design.

References

R. Balzer, "Transformation implementation: an example," *IEEE Transactions Software Engineering*, SE-7, pp. 3-14, 1981.

D. Barstow, "A perspective on automatic programming," *AI Magazine*, 5,5, 1984.

F. Frayman and S. Mittal, "Cossack: A constraint-based expert system for configuration tasks," II Int. Conf. on Appl. of AI to Engg., Boston, MA, 1987.

P. Freeman and A. Newell, "A model for functional reasoning in design," in *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pp. 621-640, 1971.

P. E. Friedland, "Knowledge-based experiment design in molecular genetics," Rep. No. 79-771, Computer Science Dept., Stanford University, 1979. (Doctoral dissertation.)

Adele E. Howe, Paul R. Cohen, John R. Dixon and Melvin K. Simmons, "Dominic: A domain-independent program for mechanical engineering design," *Artificial Intelligence in Engineering*, Vol. 1, No. 1, pp. 23-28, 1986.

Johnson, W. L. and Soloway, E., "Automatic Bug Detection," *Byte Magazine*, April, 1985.

J. Josephson, B. Chandrasekaran, J. Smith and M. Tanner, "A mechanism for forming composite explanatory hypotheses," *IEEE Transactions on Systems, Man and Cybernetics*, Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning, May/June, pps. 445-454, 1987.

Van E. Kelly, "The CRITTER system-automating critiquing of digital circuit designs," *Proceedings of the 21st Design Automation Conference*," IEEE, June, 1984.

A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, 8,1, 1977.

S. Marcus, J. McDermott and T. Wang, "Knowledge acquisition for constructive systems," IJCAI-85, pp. 637-639.

D. V. McDermott, "Circuit design as problem solving," in *AI and Pattern Recognition in CAD*, North-Holland, (Ed.), J-C. Latombe, pp. 227-245, 1978.

J. McDermott, "R1: a rule-based configurer of computer systems," in *Artificial Intelligence*, Vol. 19, No. 1, September, pp. 39-88, 1982.

G. A. Miller, E. Galanter and K. H. Pribram, *Plans and the structure of behavior*, New York: Holt, 1960.

T.M. Mitchell, L.I. Steinberg, and J. S. Shulman, "A knowledge-based approach to design," Technical report, LCSR-TR-65, Department of Computer Science, Rutgers University, January, 1985.

S. Mittal and A. Araya, "A knowledge-based framework for design," *Proceedings of AAAI-86.*

S. Mittal, C. Dym and M. Morjaria, "Pride: an expert system for the design of paper handling systems," *IEEE Computer*, 19,7, July, pp. 102-114, 1986.

A. Newell, "Reasoning, problem solving and decision processes: The problem space as a fundamental category," in R. Nickerson, ed., *Attention & Performance VIII*, Erlbaum, Hillsdale, NJ, 1980.

Charles Rich, H. Shrobe and R. C. Waters, "Overview of the programmer's apprentice," IJCAI-1979.

E. D. Sacerdoti, "A structure for plans and behavior," New York: American Elsevier, 1977.

R. Schank, *Dynamic memory: A theory of learning in computers and people*, Cambridge University Press, 1983.

R. C. Schank and R. P. Abelson, *Scripts, plans, goals, and understanding*, Hillsdale, N.J.:Lawrence Erlbaum, 1977.

Richard M. Stallman, and Gerald J. Sussman, "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis," *Artificial Intelligence*, Vol. 9, No. 2, 1977.

G. Sussman, "A computational model of skill acquisition," Ph.D thesis, MIT Math Dept., 1973.

From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task

B. CHANDRASEKARAN and ASHOK GOEL

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

January, 1988

This paper is to appear in IEEE Transactions on Systems, Man and Cybernetics, 1988. An earlier version of the paper appears in *Pattern Recognition in Practice II*, E.S. Galsema and L.N. Kanal: editors, pages 547-559, North-Holland, 1985.

From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task

B. CHANDRASEKARAN and ASHOK GOEL

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University

Abstract

We consider the very general information processing task of *classification*, and review it from the perspectives of the knowledge-based reasoning, pattern recognition, and connectionist paradigms in Artificial Intelligence, paying special attention to knowledge-based classificatory problem solving. We trace the evolution of the mechanisms for classification as the computational complexity of the problem increases, from numerical parameter setting schemes, through those using intermediate abstractions and then relations between symbols, and finally to complex symbolic structures which explicitly incorporate domain knowledge. The paper can be viewed as a bridge-building activity, describing the approaches of three different research communities to the same general task.

I. INTRODUCTION

Classification is a very general information processing task in which specific entities are mapped onto general categories. As the amount of data about the entity to be classified and the number of classificatory categories increase, typically so does the computational complexity of the task. In this paper, we review the classification task from the perspectives of the knowledge-based reasoning, pattern recognition, and connectionist paradigms in Artificial Intelligence (AI), paying special attention to knowledge-based classificatory problem solving. We trace the evolution of the mechanisms for classification as the complexity of the problem increases, from numerical parameter setting schemes, through those using intermediate abstractions and then relations between symbols, and finally to complex symbolic structures which explicitly incorporate domain knowledge. The paper can be viewed as a bridge-building activity, describing the approaches of three different research communities to the same general task. It can also be viewed as an attempt, by using the classification task as a concrete example, to give an intuitive account of how the information processing activity underlying thought necessarily evolved into complex symbolic processes in order to handle increasing complexity of problems and requirements of flexibility.

II. THE CLASSIFICATION TASK

Classification, sometimes called categorization in the cognitive science literature, as an information processing task can be functionally specified by the information it takes as input, and the information it gives as output. In its general form, the input to the classification task is a collection of data about some specific entity (*e.g.*, an object, a state, a case, or a situation), and the output is the general category (or categories) pertaining to the entity. We note that this characterization of the classification *task* as a map from specific entities to general categories makes no commitments to the *mechanism* by which the mapping is to be accomplished. Classification has been an active research issue in the knowledge-based reasoning, pattern recognition, and connectionist paradigms, though the paradigms differ in the mechanisms by which the task is performed.

A. Classification and Knowledge-Based Systems

The area of knowledge-based reasoning, though of relatively recent origin, is already a well established paradigm in AI. The essential idea of the field is to capture in computer programs, explicitly and in symbolic form, the *knowledge and problem solving methods* of human experts for selected domains and tasks. In fact, because of the central role of explicit domain knowledge of human experts, the field is often called expert systems. This is not an appropriate place to discuss the general issues of knowledge representation and problem solving in the area of knowledge-based systems, many of which remain open and active research issues. There are many expert tasks that have been successfully emulated by these systems; there are an even larger number of things that human experts do that are beyond the current state of technology for construction of knowledge-based systems. Nevertheless, when we examine the intrinsic nature of the *tasks* that knowledge-based systems perform, a surprising fact emerges: many of them solve variants of problems which are intrinsically *classificatory* in nature. We are not suggesting here that the authors of these programs *recognized* them as classification problems and used methods appropriate to the classification task, but that independent of how they were solved the problems have an intrinsically classificatory character. Let us consider some examples:

- The MYCIN system [35], in its diagnostic phase, has the task of classifying patient data onto an infectious agent hierarchy, *i.e.*, the diagnostic task is identification of an infectious agent category, as specific as possible, that pertains to the patient data.
- The PROSPECTOR system [14] classifies a geological description as corresponding to one or more mineral formation classes.
- The SACON System [3] classifies structural analysis problems into categories for each of which a particular family of analytical methods is appropriate.
- The MDX system [6], [8], [20] *explicitly* views a significant portion of the diagnostic task as classifying a complex symbolic description (the patient data) as an element, as specific as possible, in a disease classification hierarchy.

We do not mean to imply that all problems are classification problems, or that they can be usefully converted into such problems. R1 [27] and AIR-CYL [5], *e.g.*, perform different versions of the *object synthesis* problem, *i.e.*, simple versions of the design problem. Dendral [4], Internist [30] and RED [22] are different systems all performing various versions of *abductive assembly of composite explanatory hypotheses*. Chandrasekaran [7], [9], [10], has provided taxonomies of such *generic tasks*, and has identified classification as one of them. Recently, Clancey [12] has made a similar assessment of how several knowledge-based systems perform classificatory problem solving.

B. Classification and Pattern Recognition Models

The area of pattern recognition, now nearly thirty years old, represents another paradigm in AI. The classification task has been intimately associated with pattern recognition models from the very beginning of the field. In fact, in the early days of AI, the problem of recognition was *formulated* as a problem of classification, in particular one of statistical classification of pattern vectors onto one of a finite number of categories, each category characterized by some kind of probability distribution. Indeed, what started out as a practically useful formulation became so dominant that there was a need for a paper such as that by Kanal and Chandrasekaran [23] pointing out that classification is only one of the formulations for the more general recognition problem. Even when newer techniques such as syntactic techniques came into the field, the problem was still often formulated as a classification problem, this time into grammatical categories.

C. Classification and Connectionist Networks

"Neural" modeling, which predates the early perceptron models and appears to be undergoing a revival in its modern "connectionist" version, is still another paradigm in AI. The essential idea in this area is to represent knowledge as numerical weights of connections between units in a network. A variety of neural models, from linear threshold, digital networks [15], [32], to non-linear analogue architectures [21], have been developed. These models typically deal with motor or perceptual phenomena; neural networks that capture a range of complex, higher-level cognitive processes have yet to be proposed. Although our remarks are intended to be more generally applicable, in this paper we will confine our discussion only to linear threshold, digital networks in the connectionist mold in which the emphasis is on the memory and learning aspects of reasoning.

The earlier connectionist networks, *e.g.*, the perceptron model, were once viewed as devices for practical visual pattern recognition, and since the problem of pattern recognition itself was viewed as that of classification, perceptrons were really classificatory devices. The important role of classification is evident even in the more recent connectionist architectures, in which "hidden" units separate the input and the output units. Let us consider, as an example, the MBRtalk system [37], a connectionist scheme for the task of word pronunciation. It uses a numerical relaxation technique for problem solving, and a method for back propagation of corrective feedback during learning. The important point for our purposes, however, is that MBRtalk performs its task by *classifying* character substrings of the input words onto phonemes.

III. The Ubiquity of Classification

There are two things that are important to note from the above discussion: firstly, classification appears to be a rather ubiquitous information processing task, and secondly, classification has been an important research issue in the various paradigms in AI. This suggests that classification is not an artifact of any one point of view, but rather a "natural kind" of information processing task of considerable cognitive significance. Indeed, classification appears to be a powerful human strategy for organizing knowledge for comprehension and action. The human tendency to classify input entities is so strong that we often classify without necessarily being consciously aware of it, and feel we have accomplished something by merely naming entities as categories, even if we cannot do much about it. The use of classification as a strategy for knowledge organization can be found in virtually every area of human intellectual activity. In Biology, *e.g.*, taxonomic classification has long been an important methodology for organization of knowledge, and recently, mathematical techniques has been pressed into service for providing better classification in this field [36]. Some of the more recent controversies regarding evolutionary biology, *e.g.*, the traditional gradual evolutionary *vs.* the punctuated equilibrium theories, also revolve around implications of various theories of biological classification. The periodic table of chemical elements is another common classification structure in which first groups of elements and then the specific elements are identified.

A. The Computational Power of Classification

A simple computational explanation can be given for the importance of classification as an *information processing strategy*. We can think of a general task of an intelligent agent as performing actions on the world for achieving certain goals, where the right action for accomplishing a specific goal typically is a function of the relevant states of the world. In the medical domain, for example, we may view the general problem facing the physician as that of finding an appropriate therapeutic action for a given set of symptoms that describes the state of a patient and is a subset of the set of all possible symptoms. One way of mapping states of the world to actions on it might be to use a *decision table* that relates various subsets of state variables to the action variable. However, if there are n state variables $v_1, v_2, \ldots, v_n$, each of which may take on one of q values, then both the time and space complexities of mapping the states onto actions by table look-up are *$O(n.q^n)$* [17]. Thus, the table look-up approach to making decisions about actions on the world would be useful only for very small problems. In fact, the cardinality of the relevant states of the world generally is very large, *e.g.*, in the medical domain, the total number of possible states of a patient is the cartesian product of the distinct values for each of the state variables (symptoms, values from laboratory tests, other manifestations etc.). Thus, for complex, real world problems such as medical problem solving the decision table is bound to be too large for construction, storage, looking up, and modification.

The general problem of finding the right action may be solved more efficiently, however, if action knowledge can be *indexed*, not by the states of the world, but *by equivalence classes of states of the world.* A physician's therapeutic knowledge, *e.g.*, may be indexed not directly by the detailed values of the patient state variables, but by diseases, each of which can be thought of as defining an equivalence class of patient state variables. What we are suggesting here is that a *functional decomposition* of mapping states of the world to actions on it into first mapping the states onto their equivalence classes, and then using these classes for indexing the right actions often results in substantial reduction in the computational complexity of the problem since the number of equivalence classes typically is much smaller than the total number of states. The classification task corresponds to the first component in this decomposition, in which specific entities such as states of the world are mapped onto general categories which represent their equivalence classes. Medical problem solving thus may be organized first as classifying patient symptoms onto disease categories, *i.e.*, diagnosis as classification, and then indexing the therapeutic actions by the disease categories. It may not, of course, always be possible to decompose the general problem of finding the right action in such a manner; however, whenever possible, it is computationally advantageous to do so. The decomposition of mapping states of the world to actions on it is illustrated by the JESSE system [18], which supports a simple version of political decision making. JESSE first classifies the state variables describing a given situation onto situation assessment categories, and then uses these categories to index appropriate policies for action from a store of policy options.

B. Classificatory Categories

Classificatory categories represent the equivalence classes of entities that are input to the classification task. Much of human thinking is organized around classification, both in terms of acquiring new classificatory categories, and using existing categories to perform classifications, since classification provides a substantial computational advantage in solving problems. In knowledge-based systems, the classificatory categories typically are labeled symbolically, and often correspond to *concepts* in the task domain. In connectionist networks on the other hand, no labels are associated with the categories, and the categories do not necessarily correspond directly with the domain concepts. The process of creating useful classificatory categories by concept learning generally a a much harder process than using an existing classification structure. Thus, in medicine, discovery of a disease, *i.e.*, creation of a new category, is a relatively major event while diagnosis is much more routine. How these classificatory categories are created is an issue in research on learning and deep cognitive models [34]. In this paper we will deal only with the process of assigning an entity to an existing category in a classification structure.

IV. NUMERICAL APPROACHES TO CLASSIFICATION

So far we have discussed what is classification and why is it useful, but not how classification is accomplished, *i.e.*, we have presented the forms of input and output information for the classification task, and have provided an explanation for the usefulness of classification as a strategy, but have not presented any mechanism for performing the task. In the remainder of this paper we will review various knowledge-based, pattern recognition, and connectionist approaches to classification. In this section we will discuss numerical parameter setting approaches to classification. In the next section we will show how the use of intermediate abstractions reduces the computational complexity of performing the classification task, and discuss why symbols may be used to capture these abstractions. In section VI. we will discuss the use of syntactic and structural relations between symbols for classification, and in section VII. we will provide a detailed account of how complex symbolic structures that explicitly incorporate domain knowledge may be used for classification.

A. Statistical Pattern Recognition

Most early pattern recognition models used the statistical approach to classification [13] in which the object of unknown classification is represented as a multidimensional pattern vector. Each dimension of the vector represents an *attribute* of the entity, and typically is represented as a numerical variable, even though ordinals are some times used. The choice of the attributes of the entity is such that they have the potential to *distinguish* between the categories, where each category is characterized by some kind of probability distribution. In the task domain of medical diagnosis, *e.g.*, if it is desired to distinguish between diseases D_1 and D_2 and the system designer has reason to believe that symptoms s_1, s_2...s_n carry useful information for this discrimination, then often careful statistical data gathering is possible such that a *discriminant function* of the variables s_1, s_2...s_n is a very accurate classifier. When the number of dimensions is small, it is possible to design statistical classification systems that outperform human performance, since human reasoning with the same number of variables may be less efficient in information extraction. Despite the enormous intrinsic interest in the mathematical problem of designing classification algorithms in the discriminant function framework, Kanal and Chandrasekaran [24] have pointed out that the real computational power often comes from a careful choice of the attributes based on a good knowledge of the domain, rather than from the specific design of the separation algorithm.

What happens when the dimensionality of the pattern vector becomes very large, or the number of categories becomes large? When the number of categories increases, then in order to make more and more distinctions, generally the number of measurements on the entity of interest, *i.e.*, the dimensionality of the pattern vector, also needs to grow rapidly. The computational complexity of the algorithm to make the discrimination grows even more rapidly than the increasing number of dimensions, and correspondingly, the average performance, *i.e.*, the correct classification rate, deteriorates quite rapidly. Sensitivity problems become quite severe, *i.e.*, the required precision of the variables in the classification algorithm becomes impractically high. Opacity problems result, *i.e.*, it becomes increasingly hard to make any kind of statement about what attributes are playing what role in the recognition process. Szolovits and Pauker [40], discuss these and some of the other problems with probabilistic approaches to classification.

B. The Perceptron Model

Roughly in parallel with the development of statistical approaches to classification in the pattern recognition paradigm came the development of the early connectionist models of classification, specifically, the perceptron model. The perceptron architecture [31], consists of a set of input units and an output unit, each unit being a two-state, linear threshold digital device. Each unit in the input layer is connected directly to the output unit, with some numerical weight associated with each such connection. The inputs to the perceptron are points in an orthographic projection of the object to be classified, where each input unit scans some points in the projection. The output is the truth value of some predicate such as the predicate stating that the object, o_X, of unknown classification belongs to some known category, C_Y. The numerical weights associated with the connections in the network act as parameters of the network, and collectively represent the discriminant function for classification of the input object onto different categories. The output of the network is computed by a *linear combination* of the evidence that flows into the output unit *via* the connections. The perceptron architecture can be trained to "learn" the discriminant function by appropriately adjusting the weights of the connections in the network. Feedback on whether the network has reached the correct classificatory conclusion is provided by the trainer during the learning sessions. It has been shown that if the input objects are *linearly separable* then the weights of the connections will converge to the discriminant function that can correctly distinguish between the objects in finite time.

When the number of categories and the number of points scanned on the objects to be classified are small then the perceptron can be powerful classifier, at least for linearly separable objects. However, when these numbers get larger then the perceptron suffers from problems similar to those in the statistical approaches to classification. As the number of categories increases, the number of points needed to be scanned by the input units for learning the discriminant function increases, which results in a rapid increase in the number of input units. The time complexity of learning the right weights for correct classification grows even more rapidly, and correspondingly, the correct classification rate drops rapidly for a fixed number of input units. The sensitivity problem worsens, *i.e.*, even slight errors in the weights of the connections may result in large changes in the output. The opacity problem, *i.e.*, recognizing specifically which weight is playing precisely what role in the classification process, hard in the perceptron model in any case, becomes even harder. Minsky and Papert [28] discuss the computational properties of the perceptron architecture, and point out some of the problems with it.

V. USE OF INTERMEDIATE ABSTRACTIONS IN CLASSIFICATION

The above discussion shows that while numerical parameter setting schemes may lead to powerful classifiers for small problems, the complexity of the separation algorithm becomes impractically high as the number of classificatory categories increases. The problem here lies not so much in the specific choice of one discriminant function over another, but in the fact that these approaches seek to *directly* map the input entity onto classificatory categories. Indeed, similar complexity problems arise for *all* approaches that perform classification by directly mapping specific entities onto general categories. Let us consider, as another example of such direct classification, the method of discrimination tree traversal for medical diagnosis. Again, let the input be characterized by *n* state variables, $s_1, s_2, \ldots, s_n$, each of which can take on one of *q* values. The state variables are organized in a tree in which the top node corresponds to some state variable s_1 and has q branches coming out of it, one for each of the q possible values that s_1 may take. The branches lead to q different nodes, each of which corresponds to some s_2 and has q branches coming out of it. This organization is repeated until all the state variables have been represented on the tree. Each of the q^n branches coming out of the q^{n-1} nodes at the n^{th} level leads to one of a finite number of disease categories, D_1, D_2,..., D_m. The time and space complexities for classification by discrimination tree traversal are given by $O(n)$ and $O(q^n)$, respectively [17]. Clearly, for complex, real world problems, where the number of classificatory categories typically is large, the proposition of directly mapping input entities onto classificatory categories is quite futile.

What, then, can be done when the number of classificatory categories is large? Let us consider, as an example, the problem of automatic reading of texts in some language that consists of a large number of words. Intuitively, one would think that first recognizing characters (or perhaps substrings of characters) in the words, and then recognizing word themselves would be computationally more attractive. The words (or perhaps word phrases) may be later used in understanding complete sentences in the language. In this approach, instead of performing classification by a direct mapping from the input entity onto the categories, intermediate abstractions are first constructed, the entity of unknown classification mapped onto these abstractions, which are then used as inputs to a higher-level classification process. What we are suggesting here is a *conceptual decomposition* of the classification process onto *hierarchically organized intermediate abstractions*. Such a conceptual decomposition makes the classification process more efficient, as we will see a little later.

A. Signature Tables

In order to make the notion of conceptual decomposition of the classification process into hierarchically organized intermediate abstractions more explicit, let us consider *evaluation functions* in game playing, *e.g.*, playing chess, as another example of classification. These functions usually yield a number which is a measure of the "goodness" of the board. For most purposes, effective use of this information can be made if the goodness is classified into one of a small number of categories. One of the first forms proposed for the evaluation functions was a linear polynomial of attributes of the board, where both the attributes and their weights were chosen in consultation with domain experts. Later, in order to take into account interactions between the variables in the evaluation function, higher order polynomials were proposed. This of course resulted in a fairly rapid increase in the complexity of the function: if r^{th} order interactions between the attributes were to be included, and the number of attributes is *n*, then the number of terms was of the order of n^r. Samuel's *signature tables* [33] provided a solution which exemplifies the use intermediate abstractions in classification. For the purposes of our discussion, Samuel's method can be described as follows:

1. Identify groups of attributes such that on the basis of domain knowledge there is reason to believe that they contribute to an intermediate abstraction that can be used to construct the desired classification, which in this case is a measure of the goodness of the board. The number of attributes in each group is kept small, and the attributes in a group may have some dependencies and interactions, in order to capture which polynomial terms were included in the more traditional evaluation functions. The abstractions typically correspond to the concepts in the task domain, *e.g.*, in chess, "defensibility of king" and "material advantage" may be such intermediate concepts, each of which can be estimated by a small subset of board attributes, while the final decision about the goodness of a board configuration may be made in terms of these intermediate abstractions.

2. Find a method of *classifying* the desirability of these intermediate concepts into a small number of categories from the values of the attributes in each group. The exact method for this classification is not especially important here, though Samuel proposed a specific mechanism for it. The essence of his mechanism is a mapping from a multidimensional vector, each component of which can only take on one of a small number of distinct values, to a symbolic abstraction, which can also take on one of a small number of distinct values. This mapping may be performed by a simple table look-up for example.

3. The outputs of the classifiers for each group can themselves be thought of as *qualitative* attributes at the next level of abstraction. These attributes can be then grouped and abstracted into higher level concepts, and the process repeated as many times as necessary, with only a small number of attributes in a group at any level, until the top-level concept is a classification of the "goodness" of the board.

Let *n* denote the total number of attributes at the lowest level of abstraction. Let us assume that the number of attributes in each group at any level in the hierarchy of abstractions is smaller than some small, constant, upper bound n_0 (an assumption allowed in the signature table method), and further, that the groups of attributes at any level are disjoint. Then both the time and space complexities are *O(n)* [17]. Even if a few attributes at some level are used in more than one group of attributes, which sometimes is the case, and in which case the time complexity would be somewhat worse than linear in *n*, clearly, the use of intermediate abstractions in classification yields substantial computational savings. Again, we are not suggesting that such conceptual decomposition of the classification process into hierarchically organized intermediate abstractions is always possible, but that, whenever possible, it is computationally advantageous to do so.

B. Hidden Units in Connectionist Networks

The computational power of using intermediate abstractions is evident from the fact that a major difference (perhaps *the* major difference) between modern connectionist networks and the perceptron model, is that the former provide mechanisms for capturing intermediate abstractions. In the perceptron model, since the input units were connected directly to the output unit, there was no representational mechanism to capture intermediate abstractions, and classification was performed by directly mapping input objects onto categories. Modern connectionist networks, on the other hand, contain hidden units between the input and the output units, thus providing a mechanism for representing intermediate abstractions as patterns of activity over the hidden units. The notion that the real role of the hidden units is to somehow capture these abstractions becomes clear from the following observation: in most connectionist schemes, such as the one for learning the past tenses of English language words [32], the *number of hidden units in the network is critical to its performance.* When the number of hidden units is too small then the problem is overconstrained and there is not enough structure to capture all the needed abstractions, as a result of which the performance of the network deteroriates markedly; and when the number of hidden units is too large then the problem is underconstrained and generalizations to the abstractions are not possible, again resulting in a marked deteroriation in the network performance. One method of handling these sensitivity problems is to make the number of hidden units a parameter of the architecture, and then experiment with the value of this parameter until the number of hidden units in the network is just right.

The real computational power of modern connectionist networks is thus based on the use of intermediate abstractions, which is an important reason for the resurgence of the connectionist paradigm in AI more than a decade after Minsky and Papert had showed the inadequacies of the perceptron model. Classification in connectionist architectures is accomplished by first mapping the input entity onto classificatory abstractions, and then mapping these abstractions onto output categories. Moreover, as in Samuel's work on signature tables for game playing programs, in modern connectionist networks the intermediate abstractions can be organized hierarchically. Indeed, for large scale connectionist networks, where the number of classificatory categories and intermediate abstractions may be very large, hierarchicalization of abstractions is an important method for dealing with the complexity of learning classificatory categories and intermediate abstractions [2].

C. Symbols and Abstractions

While the intermediate abstractions are represented as patterns of activity over the hidden units in connectionist networks, there is simpler way of capturing these abstractions: by means of discrete *symbols.* The representation of abstractions by symbols entails a trade off between the precision of numbers, with the concomitant problems of complexity, sensitivity, and opacity, for the *simplicity, flexibility, and perspicuity* of symbols. Often numbers are too precise for the task at hand, and robust symbolic hierarchical abstractions of the appropriate kind can capture almost all of the relevant information. These advantages of representing abstractions by symbols have been demonstrated most recently by Lehnert [25]. She has constructed a connectionistically inspired system, called PRO, for the task of word pronunciation, the same task that is performed by the entirely connectionist MBRtalk system. The main difference between the two approaches lies in that the PRO system uses symbols for capturing intermediate abstractions in the classification of character substrings of words. While PRO appears to perform at least as well the MBRtalk system, it is simpler, smaller, more robust, and more perspicuous. We are not suggesting that intermediate abstractions are entirely neutral to the underlying architecture of implementation and representing abstractions symbolically is necessarily right for all tasks. Chandrasekaran *et al.* [11] provide an analysis of the interaction between the abstractions needed for problem solving and the architecture for their implementation, and suggest that connectionist schemes may be well suited for simple forms of pattern matching and data retrieval, and for low-level parameter learning. However, for capturing higher level cognitive processes the advantages of using symbols for representing abstractions are just too important.

VI. USE OF RELATIONS BETWEEN SYMBOLS FOR CLASSIFICATION

After about a decade of work on statistical classification in the pattern recognition paradigm, during which work on classification in the perceptron and the symbolic paradigms was going on roughly in parallel, Narasimhan [29] proposed a *syntactic approach* to pattern classification. The idea was to describe categories of patterns not in terms of probability distributions in multidimensional spaces, nor in terms of intermediate abstractions that can be captured symbolically, but in terms of *relations between symbols*, much as grammatical categories are described in linguistic analysis. The idea of syntactic pattern recognition is really a special case of the more general notion of *structural relations* for describing classificatory categories. Thus, even when the idea of *syntax* is not appropriate --- it is doubtful that the notion of a picture grammar really is as general for the domain of visual objects as it appears from a purely formal perspective --- the notion of structural relations for characterizing categories may still be applicable. We note that the ability to describe a category in terms of relations is a move towards *descriptions* as the basis for category characterization.

The major research directions in pattern recognition for capturing structural relations generally were *formal*, *i.e.*, they used some or the other mathematical system within which theorems about relationships between categories may be provable regarding the classification performance. In fact, this was the major reason for the original emphasis on syntactic methods, since there was a well developed theory of formal grammars already available. This emphasis on formalisms led to two constraints: firstly, often an attempt was made to force the available formalisms to fit the pattern recognition problem, generally with unsatisfactory results; and secondly, because human classification performance was more heuristic in nature, restricted formalisms could capture the quality of human performance only fleetingly.

It is interesting to note that in connectionist schemes also classification is based on structural relations between intermediate abstractions, even though the abstractions are represented by patterns of activity over hidden units instead of being captured symbolically. The structural relations themselves are represented by connections of various types between the hidden units. Thus, in the MBRtalk system, the connectionist scheme for the task of word pronunciation, classification of the input words is based on the "syntactic relations" between the non-symbolic classificatory abstractions [37].

With the introduction of syntactic/structural relations between intermediate abstractions the progression of approaches to classification becomes

numbers ---> abstractions (symbols) ---> relations.

Now, if one is to use relations between symbolic attributes as the basis of category characterization, then why restrict oneself to *syntactic* relations? Why not bring the full power, to the extent possible or necessary, the *semantics* of the classificatory categories? Asking this question prepares the way for the next step in the progression of approaches to classification.

VII. KNOWLEDGE-BASED APPROACHES TO CLASSIFICATION

It is clear that each AI paradigm emphasizes different issues and poses them in a different language, *e.g.*, the pattern recognition paradigm raises issues such as those of discriminant functions, probability distributions, and error rates, while the connectionist paradigm raises issues such as those of weights of connections, hidden units, and parameter learning. Similarly, the knowledge-based reasoning paradigm focuses on the issues of how to *represent* knowledge in symbolic form, how to *organize* and *access* this knowledge, how to *use* this knowledge for solving problems, and how to *control* the problem solving process. The knowledge-based approaches to the classification task attempt to answer these questions for classificatory problem solving. In this section, we will describe *hierarchical classification* [6], [20] as an example of knowledge-based approaches to classification, using the task domain of medical diagnosis for illustration.

A. Hierarchical Classification

In hierarchical classification, domain knowledge is organized as a hierarchical collection of categories, each of which has knowledge that helps it determine its relevance to the input case of unknown classification. A fragment of the classification hierarchy for medical diagnosis might be as shown in Figure 1. Each category in the diagnostic classification hierarchy is a diagnostic *concept* of potential relevance to the case at hand. More general concepts (*e.g.*, LIVER) are higher in the hierarchy, while more particular ones (*e.g.* HEPATITIS) are lower in the structure.

The total diagnostic knowledge is *distributed* over the conceptual categories in the hierarchy. Each concept has "how-to" knowledge for simple evidential reasoning in the form of several clusters of *diagnostic rules*: confirmatory rules, exclusionary rules, and perhaps some recommendation rules. These production rules are of the form: <pattern> -----> <evidence>, *e.g.*, "If the value of SGOT is high then add *n* units of evidence in favor of cholestasis", where *n* is some small integer. The number of rules in any one cluster is kept small, and the evidence for confirmation and exclusion is suitably weighted and combined to arrive at a conclusion to establish or reject the relevance of the category to the case, or perhaps to suspend the decision making if there is not sufficient data to make a decision at the present time. The recommendation rules are optimization devices whose discussion is not necessary for our current purpose. What is important here is that when a concept in the classification hierarchy is properly

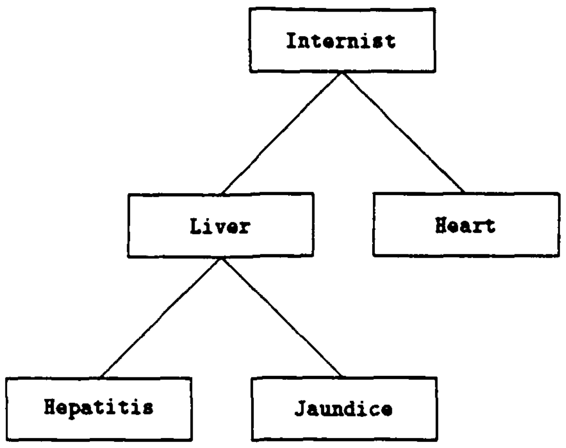


Figure 1: Fragment of a diagnostic classification hierarchy

invoked, a small, body of knowledge relevant for decision making comes into play.

The *control problem* in hierarchical classification can be stated as "which conceptual category should be considered at what point in the problem solving?". In general, we would like to use domain knowledge to achieve computational efficiency by considering only a subset of all categories. Similarly, we would like to consider categories which are more promising ahead of others. The control regime natural to hierarchical classification is top-down and can be characterized as *establish-refine*. Starting from the root node, each concept first uses its knowledge to establish or reject itself for relevance to the entity to be classified. If it succeeds in establishing itself, then it attempts refinement by *sending messages* to its subconcepts who repeat the establish-refine process. If, on the other hand, the concept rejects itself, then all its subconcepts are automatically ruled out leading to a pruning of the hierarchy. The idea is to establish a conceptual category, as specific as possible, that is relevant to the input entity. Let us consider the case of a patient suffering from hepatitis as an example. Given data about this patient, first INTERNIST would establish that there is in fact a disease, and send messages to LIVER and HEART for refinement as shown in Figure 1. Then LIVER would establish that the disease is a liver disease, and send messages to HEPATITIS and JAUNDICE for refinement, while HEART would reject the hypothesis that the patient is suffering from a heart disease. Next, HEPATITIS would establish the disease as hepatitis while JAUNDICE would rule out the hypothesis that the disease is jaundice. Thus each concept makes decisions about its relevance to the patient data in the *context* of the decisions made by its superconcepts. Sticklen *et. al.* [38] discuss the control issues in classificatory diagnosis in detail.

The problem solving in this approach to classification is *distributed*. The conceptual structures in the hierarchy are not a static collection of knowledge; instead, they are active problem-solving agents. Each of them has knowledge only about establishing or rejecting the relevance of a conceptual category, and communicates with others by passing messages. The entire ensemble of these semi-autonomous problem solving agents cooperates to perform the classification task. Goel *et al.* [19] have shown how the concurrency inherent in hierarchical classification can exploited on a distributed memory, message passing architecture.

We note that hard probability numbers are nowhere used in diagnosis by hierarchical classification; what each problem solving agent computes are *qualitative* belief measures: "definitely present", "likely present",..."definitely absent". Moreover, the computation of the qualitative values is *localized* rather than based on some global probability calculus; each agent computes the qualitative measure for *its* concept using only its own knowledge but in the context of its superconcepts. Medical diagnosis appears to be an instance of the class of problems in which a numerical approaches, such as statistical pattern recognition, would have significant computational problems. In addition, it would pose considerable difficulty in acquiring knowledge in terms of probability distributions, at least for problems of large degree of complexity, while knowledge in the form required by hierarchical classification is often directly available from domain experts.

At our research laboratory we have used the hierarchical classification methodology to construct MDX [6], [8], [20], a medical diagnostic system for a class of liver diseases in internal medicine. The number of state variables, such as symptoms, signs, and laboratory values, describing a typical case that MDX can handle is in the hundreds, and the number of distinct conceptual categories in its diagnostic hierarchy is also close to hundred. MDX is a complex system that has been tested on a number of real world cases with a high match between its conclusions and that of human specialists. Recently, a more sophisticated version of the MDX system, called MDX2 [39], has been constructed in our laboratory.

Several concerns ought to be noted before using the hierarchical classification methodology to build knowledge-based classificatory problem solvers:

1. Not all classification problems are necessarily solved as hierarchical classification problems.

   Hierarchical classification requires that concepts in the task domain be available at several different levels of abstraction. While there are many real world domains that do satisfy this condition, not every domain need have this characteristic. There are other systems that perform classification, but without using the hierarchical point of view [1]. However, it may be better to use hierarchical classification whenever possible for reasons of computational efficiency. Let *m* be the number of categories at the *leaf nodes* of the classification hierarchy. Since the desired classification generally is one of these *m* categories, the time complexity of non-hierarchical classification is $O(m.t)$, where t is the time complexity of finding the relevance of a single category to the entity of unknown classification. If the number of state variables is n, and single category classification is performed using the signature table approach discussed earlier, then t is $O(n)$. In case of hierarchical classification, in the best case when all but one branch at each node in the hierarchy are ruled out, the time complexity is $O(log(m).t)$; and in the worst case, when every branch at each node is traversed, the time complexity is $O(m.t)$. Goel *et al.* [17] provide details of the complexity calculations for classificatory reasoning. It is clear, however, that even in the worst case, the complexity of hierarchical classification is no worse than the complexity of non-hierarchical classification, and the choice between them really depends on whether it is possible to construct a classification hierarchy in the task domain of interest.

2. The entity to be classified may have several leaf node categories simultaneously relevant to it, rather than just one leaf node category. In medical diagnosis, *e.g.*, a patient may have both "cirrhosis" and "portal hypertension" (which in the domain of liver diseases might be two of leaf nodes in the classification hierarchy), and in addition, the two diseases may be causally related. Such a situation is not uncommon in other domains as well, *e.g.*, in character recognition, the pattern to be classified may consist of be two characters touching each other rather than one single character. The hierarchical classification framework clearly can deal with such situations.

3. The classification hierarchy may be a "tangled" hierarchy, *i.e.* some concepts in the hierarchy may have more than one superconcept. Such a hierarchy may be "untangled" in the hierarchical classification framework by storing a copy of the concept in each tangled branch. This introduces redundancy in the storage of domain knowledge by the classification agent.

4. In general, multiple classification hierarchies may exist in the task domain, *e.g.*, in medical diagnosis there may be one classification hierarchy for infectious diseases, and another for liver diseases. In addition, the same category may exist in more than classification hierarchy, e.g., viral hepatitis is a conceptual category in the infectious disease hierarchy as well as in the liver disease hierarchy. This involves *coordination* among the classifications reached by the different classification modules. The MDX2 system contains several classification hierarchies, and provides a mechanism for handling such interactions between them.

5. The problem task may require not only classification of entities onto categories, but other problem solving *types* as well, *e.g.*, the diagnostic task often is functionally decomposable into the generic tasks of *knowledge-directed data abstraction*, and *abductive assembly of explanatory hypotheses* in addition to that of classification [9], [10]. This involves coordinating the actions of various problem solving modules performing different generic tasks and cooperatively solving diagnostic problem. The MDX system [8] contained modules for hierarchical classification and knowledge-directed data abstraction and provided mechanisms for communication between them. The MDX2 system [39] contains modules for knowledge-directed data abstraction and abductive assembly of explanatory hypotheses in addition to several hierarchical classification modules, and provides mechanisms for handling interactions between them.

6. The conceptual structure mechanism used in hierarchical classification is only one of the several possible methods for determining the relevance of a specific category to the entity of unknown classification. In the DART system [16], *e.g.*, the decision about the match of the category to the input data is done by using theorem-proving techniques. Alternatively, the classification category agents may make their decisions based on a causal knowledge of the domain [34]. The MDX2 systems uses such causal knowledge to derive the conceptual structure needed for category classification. In simple cases, it may be possible to use statistical pattern recognition methods for this purpose. Connectionist networks may be especially appropriate for the pattern matching operations required in simple evidential reasoning [11]. The point is that *how* the hypotheses are evaluated is somewhat independent of the flow of control for the classificatory task as such, even though for complex problems, a rich knowledge structure will be called for to make the decision about

how well a specific category matches the data for the case in hand.

VIII. CONCLUSIONS

We have noted that classification appears to be an ubiquitous information processing task underlying human thought processes. The reason for this is the significant computational advantages that arise from indexing stored action knowledge over *equivalence classes of the states of the world* rather than over the states of the world themselves. We have taken the reader through a progression of approaches to classification:

numbers ---> abstractions (symbols) ---> relations ---> knowledge structures.

Each stage in this progression gave added power in controlling computational complexity by matching the structure of the classifier to that of the task. At the knowledge level, the computational power comes from *task-specific control regimes* controlling access to *appropriate chunks of domain knowledge*. We motivated the discussion by using classificatory diagnosis as an example in various places, but the ideas are applicable more generally.

This paper can be viewed as a bridge-building activity between three research paradigms in AI: knowledge-based reasoning, pattern recognition, and connectionism. Classification has been a major concern in pattern recognition, and an important task performed by most knowledge-based systems as well as by many connectionist networks. Thus, the classification task provides a good place to understand some of the distinctions between the three research paradigms. For well-constrained classification problems with relatively small number of categories, the numerical functions and measures used in pattern recognition models and connectionist networks typically can provide powerful classifiers which often outperform human experts by extracting the last trace of information that discrete symbolic processes can only approximate. On the other hand for complex problems involving many variables and categories the symbolic knowledge-based approach trades off the optimality of the best functions in pattern recognition and in connectionism for computational tractability and better matching with human knowledge in the task domain. Our own research lies in the knowledge-based reasoning paradigm. Our approach has been to identify *generic tasks* other than that of classification, but with the similar characteristic of being a building block for intelligence. Chandrasekaran [7], [9], [10] provides an account of the repertoire of generic tasks that we have identified so far.

Many of the points made in this paper transcend the particular task of classification. In that sense, this paper can be thought of as an attempt to show the need for the emergence of symbolic structures for complex information processing transformations on representations. Cybernetics showed the power and usefulness of feedback and stability in understanding many control and communication problems. However, classical control theory is expressed in terms of numerical measures and functions. Learning and control in this framework involves parameter modification and signal propagation. The space over which parametric changes and numerical signals can provide control is quite limited. Symbolic models of the world provide greater leverage for change and control and still keep computational costs under control. Thus in biological information processing, symbolization seems to have occurred very early in evolution; Lettvin *et al.* [26] provide an account of how the early visual processing of the frog is symbolic. Once symbols were available as the language in which to perform information processing, thought eventually evolved into more and more complex symbol structures. Thus the discussion in this paper can be viewed as an intuitive account of the emergence and power of symbolic structures for complex information processing activities.

Acknowledgments

We are thankful to Dean Allemang for his comments on an earlier version of this paper. We are grateful for the support of the Defense Advanced Projects Research Agency, RADC Contract (F30602-85-C0010), and the Air Force Office of Scientific Research, grant (87-0090), during the preparation of this paper.

References

[1] J.S. Aikins. "Prototypical Knowledge for Expert Systems". *Artificial Intelligence* 20(2):163-210, 1983.

[2] D.H. Ballard. "Modular Learning in Neural Networks". In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1987, pages 279-284.

[3] J. Bennet and R. Engelmore. "SACON: A Knowledge-based Consultant for Structural Analysis". In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, 1979, pages 47-49.

[4] B. G. Buchanan and E.A. Feigenbaum. "Dendral and Meta-Dendral: Their Applications Dimension". *Artificial Intelligence* 11(1-2):5-24, 1978.

[5] D.C. Brown and B. Chandrasekaran. "Knowledge and Control for a Mechanical Design Expert System". *IEEE Computer Magazine* 19(7):92-100, 1986.

[6] B. Chandrasekaran, S. Mittal, F. Gomez, and J.W. Smith. "An Approach to Medical Diagnosis Based on Conceptual Structures". In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, 1979, pages 134-142.

[7] B. Chandrasekaran. "Towards a Taxonomy of Problem-Solving Types". *AI Magazine* 4(1):9-17, 1983.

[8] B. Chandrasekaran and S. Mittal. "Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems". In *Advances in Computers*, M. Yovits: editor, Academic Press, New York, 1983, pages 217-293.

[9] B. Chandrasekaran. "Generic Tasks in Knowledge-based Reasoning: High-Level Building Blocks for Expert System Design". *IEEE Expert Magazine* 1(3):23-30, 1986.

[10] B. Chandrasekaran. "Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks". In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987, pages 1183-1192.

[11] B. Chandrasekaran, A. Goel, and D. Allemang. "Connectionism and Information Processing Abstractions: The Message Still Counts More Than the Medium". To appear in *AI Magazine*, 1988.

[12] W. J. Clancey. "Heuristic Classification". *Artificial Intelligence* 27(3):289-350, 1985.

[13] R.O. Duda, and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley, New York, 1973.

[14] R.O. Duda, P.E. Hart, P. Barret, J. Gasching, K. Konolige, and R. Reboh. "Development of the Prospector Consultation System for Mineral Exploration". Technical Report, SRI International, Menlo Park, California, 1979.

[15] J.A. Feldman and D.H. Ballard. "Connectionist Models and Their Properties". *Cognitive Science* 6:205-254, 1982.

[16] M.R. Genesereth. "Diagnosis Using Hierarchical Design Models". In *Proceedings of the Second National Conference on Artificial Intelligence*, 1982, pages 278-283.

[17] A. Goel, N. Soundararajan, and B. Chandrasekaran. "Complexity in Classificatory Reasoning". In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1987, pages 421-425.

[18] A. Goel, B. Chandrasekaran, and D. Sylvan. "JESSE: An Information Processing Model of Political Decision Making". In *Proceedings of the Third Expert Systems in Government Conference*, 1987, pages 78-87.

[19] A. Goel, J.R. Josephson, and P. Sadayappan. "Concurrency in Abductive Reasoning". In *Proceedings of the DARPA Knowledge-based Systems Workshop*, 1987, pages 86-92.

[20] F. Gomez, and B. Chandrasekaran. "Knowledge Organization and Distribution for Medical Diagnosis". *IEEE Transactions on Systems, Man, and Cybernetics* 11(1):34-42, 1981.

[21] J.J. Hopfield and D.W. Tank. "Neural Computation of Decisions in Optimization Problems". *Biological Cybernetics* 52:141-152, 1985.

[22] J.R. Josephson, B. Chandrasekaran, J.W. Smith, and M.C. Tanner. "A Mechanism for Forming Composite Explanatory Hypotheses". *IEEE Transactions on Systems, Man, and Cybernetics* 17(3):445-454, 1987.

[23] L. Kanal and B. Chandrasekaran. "Recognition, Machine Recognition, and Statistical Approaches". In *Methodologies of Pattern Recognition*, Academic Press, New York, 1969, pages 317-332.

[24] L. Kanal and B. Chandrasekaran. "On Linguistic, Statistical, and Mixed Patterns for Pattern Recognition". In *Frontiers of Pattern Recognition*, Academic Press, New York, 1972, pages 163-192.

[25] W. Lehnert. "Case-Based Problem Solving with a Large Knowledge Base of Learned Cases". In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1987, pages 301-306.

[26] J. Lettvin, H. Maturana, H., W.S. McCulloch, and W. Pitts. "What the Frog's Eye Tells the Frog's Brain". in *Proceedings of the IRE*, 1959, 47:1940-1951.

[27] J. McDermott. "R1: A Rule-Based Configurer of Computer Systems". *Artificial Intelligence*, 19(1):39-88, 1982.

[28] M. Minsky and S. Papert. *Perceptrons*, Expanded Edition, MIT Press, Cambridge, 1988.

[29] R. Narasimhan. "Labeling Schemata and Syntactic Description of Pictures". *Information and Control* 7:151-179, 1964.

[30] H.W. Pople. "Heuristic Methods for Imposing Structure on Ill-Structured Problems". In *Artificial Intelligence in Medicine*, P. Szolovits: editor, Westview Press, Boulder, Colorado, 1982, pages 119-190.

[31] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1962.

[32] D.E. Rumelhart and J.L. McClelland. "On Learning the Past Tenses of English Verbs". In *Parallel Distributed Processing, Volume 1*, Rumelhart, McClelland and the PDP Research Group: editors, MIT Press, Cambridge MA, 1986.

[33] A.L. Samuel. "Some Studies in Machine Learning Using the Game of Chequers II: Recent

Progress". *IBM Journal of Research and Development* 11(6):601-617, 1967.

[34] V. Sembugamoorthy and B. Chandrasekaran. "Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems" In *Experience, Memory, Reasoning,* J. Kolodner and C. Reisbeck: editors, Lawrence Earlbaum, Hillsdale N.J., 1986, pages 47-73.

[35] E.H. Shortliffe. *Computer-based Medical Consultations: MYCIN.* Elsevier/North-Holland, 1976.

[36] R.R. Sokal and P.H.A. Sneath. *Principles of Numerical Taxonomy.* Freeman, San Francisco, 1963.

[37] C. Stanfill and D. Waltz. "Toward Memory-based Reasoning". *Communications of the ACM* 29(12):1213-1228, 1986.

[38] J. Sticklen, B. Chandrasekaran, J.R. Josephson. "Control Issues in Classificatory Diagnosis". In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* 1985, pages 300-306.

[39] J. Sticklen. "MDX2: An Integrated Medical Diagnostic System". PhD. Dissertation, Department of Computer and Information Science, The Ohio State University, 1987.

[40] P. Szolovits and S.G. Pauker. "Categorical and Probabilistic Reasoning in Medical Diagnosis". *Artificial Intelligence* 11:115-144, 1978.

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
April 1987

DATA VALIDATION DURING DIAGNOSIS, A STEP BEYOND TRADITIONAL SENSOR VALIDATION

B. Chandrasekaran and W. F. Punch, III
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210

To appear in *Proc. Sixth National Conference on Artificial Intelligence.*

Data Validation During Diagnosis, A Step Beyond Traditional Sensor Validation.

B. Chandrasekaran and W.F. Punch III[1]

Laboratory for Artificial Intelligence Research
The Ohio State University

Abstract

A well known problem in diagnosis is the difficulty of providing correct diagnostic conclusions in light incorrect or missing data. Traditional approaches to solving this problem, as typified in the domains of various complex mechanical systems, validate data by using various kinds of redundancy in sensor hardware. While such techniques are useful, we propose that another level of redundancy exists beyond the hardware level, the redundancy provided by expectations derived during diagnosis. That is, in the process of exploring the space of possible malfunctions, initial data and intermediate conclusions set up expectations of the characteristics of the final answer. These expectations then provide a basis for judging the validity of the derived answer.' We will show how such expectation-based data validation is a natural part of diagnosis as performed by hierarchical classification expert systems.

1. Introduction

Diagnosis is the process of mapping system observations into zero or more possible malfunctions of the system's components. Most of the work in AI in diagnosis assumes the observations given to an expert system are reliable. However, in real-world situations, data is often unreliable and real-world diagnostic systems must be capable of taking this into account just as the human expert must. In this paper, we will discuss a knowledge-based approach to validation that relies on *diagnostic expectations* derived from the diagnostic process itself to identify possible unreliable data points.

Present-day aids for human experts performing diagnosis attempt to validate data before diagnosis begins. In the domains of various complex mechanical systems (Nuclear Power Plants, Chemical Manufacturing Plants, etc.), such aid is based on the concept of hardware redundancy of sensors. Each important system datum (pressure, temperature etc.) is monitored with a *number* of hardware sensors providing a redundancy of information from which a composite, more reliable value is extracted. Based on this hardware redundancy, a number of techniques were developed to validate a datum's value:

1. Providing multiple sensors of the same kind to monitor a datum. Loss of one sensor therefore does not preclude data gathering and any disagreements among the sensors can be resolved statistically. For example, to measure the temperature of a chemical reaction, multiple temperature sensors could be used in the reactor and their statistical average given as the overall temperature value.
2. Providing different kinds of sensors to monitor a datum. This situation provides the same redundancy as (1) as well as minimizing the possibility of some kinds of common fault problems. That is, certain events that inactivate one sensor type will not affect sensors of a different type. Continuing with the example of (1) above, half of the sensors might be thermocouples while the other half might be mechanical temperature sensors.
3. Using sensors in several different locations to infer a datum value. In this situation, data values are monitored both directly and inferred from other system data based on well-established relationships. For example, while the temperature of a closed vessel may be directly monitored, it can be inferred from the measurement of the pressure using the $PV = nrT$ equation.

Such hardware redundancy allows some data validation, but some limitations for this approach do exist:

1. The expense of installing and maintaining multiple sensors for each important datum greatly increases the cost of the mechanical system.
2. Common fault failures still happen, despite cautions mentioned above, especially as the result of severe operation malfunctions.
3. Human operators and engineers resolve many such diagnostic problems despite incorrect and even absent data. In other words, human experts are more tolerant of *bad* data whether it has been validated or not.

Therefore, while hardware redundancy does solve part of the problem, more sophisticated techniques are required to complete the job.

The following simple example will help in examining point (3) and other ideas[2]. Consider the mechanical system diagrammed in Figure 1 with data values indicated in Figure 2. It is a closed vessel with two subsystems, a cooling system and a pressure relief system. The vessel is a reactor which contains some process (nuclear fission, chemical reactions, etc.) that produces both heat and pressure. The data values of Figure 2 indicate that the temperature of the reactor vessel is above acceptable limits.

[1] We gratefully acknowledge the support of grants from the Air Force Office of Scientific Research, AFOSR-82-0255, and the National Science Foundation, CPE-8400840

[2] Note that the ideas presented here have been used on more complicated real-world systems [5, 7]. This example has been condensed from them for expository clarity.

Assume for the example that two possible causes of the high reactor temperature exist: either the cooling system has failed or the pressure relief system has failed and the added heat has overpowered the functioning cooling system. Given the sensor readings, what would be the diagnostic conclusion? The data conflict is the *normal* pressure and cooling system readings and the *abnormal* pressure relief system readings The failure of the pressure relief system is plausible, data indicates its failure and no other system failure, but such a failure expects the pressure to be high! The step to take is to assume that both the pressure relief system failed and the pressure sensor is incorrect.

The process shown above demonstrates data validation at a higher level than that of simple sensor hardware validation. In the example, the pressure system has failed despite the lack of a high pressure datum. However, there is other strong evidence[3] that the pressure system has indeed failed. The human reasoner *expects* the pressure datum to be high since the preponderance of other data indicate a malfunction. That is, the human reasoner in pursuing likely diagnostic conclusions discovers a plausible diagnostic conclusion that meets all but (in this case) one expectation. The important points to note are that:

1. A diagnostic conclusion can and should be made based on the preponderance of other evidence.
2. The datum value that does not meet expectation should be questioned and further investigation of its true value made.

Note that this process involves redundancy, not at the level of sensor hardware, but at the level of *diagnostic expectation*. This is a redundancy of information that allows questioning (and subsequent validation) of data *based on multiple expectations of* diagnostic conclusions. If a conclusion is likely, but not all of its expectations are met, then those now questionable values are *investigated* by more computationally expensive techniques.

Such expectations can be the result of one of a number of processes. Deep models can provide information on expected data patterns for any diagnostic conclusion. From this information, judgments on the reliability of any of the actual data values can be made. Information provided from such deep models can be incorporated into compiled structures that can also provide information on data reliability. Finally, the expert himself can provide the information on data reliability to the diagnosis system based on his expert judgment of the particular diagnostic process, in effect acting as the deep model for the system.

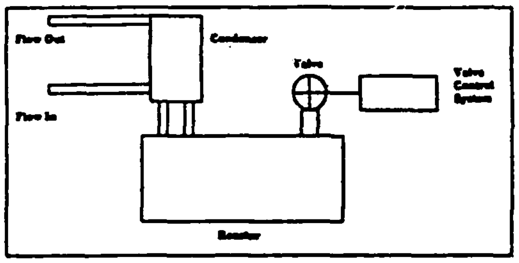


Figure 1: An Example Mechanical System

In this paper we will discuss compiled diagnostic systems that deal with conflicting datum at the level of diagnostic expectation indicated above. These are diagnostic systems that make conclusions based on diagnostic knowledge and some judgment on the validity of the data provided. In particular, we will show how

[3]In the example, this evidence is that there is a failure of the relief valve system which is part of the pressure system.

Variable	*Status*
Temperature	High
Pressure	Normal
Condenser	All sensors Normal
Cooling Water Flow System	All Sensors Normal
Relief Valve	Sensors Indicate Malfunction
Valve Control System	All sensors Normal

Figure 2: Sensor Values for the Example

redundancy of diagnostic expectation is a natural extension to the hierarchical classification diagnostic model.

2. Diagnosis as a Hierarchical Classification Task

Significant interest has recently been directed towards understanding problem solving behaviors (diagnosis, planning, design) from the viewpoint of Information Processing Strategies. For example, Clancey [4] has shown that MYCIN is a species of classification problem solving activity. Earlier, in our work on MDX [2], we explicitly identified *hierarchical classification* as a strategy useful for some classes of diagnostic problem solving.

Diagnosis as a classification problem solving task is a *matching of the data of the problem against a set of malfunctions* (i.e., diseases, system failures, etc). If the present data are classified as a known malfunction, then the diagnosis is completed. Note that this is a *compiled* approach to diagnosis since it requires that the possible malfunctions and knowledge about how to establish those malfunctions be pre-enumerated. Other less well defined problems require a deeper model that rely on first principles (physics, chemistry, etc.) and an intimate understanding of the system at hand[4]. The rest of this section discusses the basic ideas behind hierarchical classification (See [6, 2] for details).

The malfunctions (diseases, failures) possible in the system are organized hierarchically. Typically, this hierarchy reflects a *system:sub-system or function:sub-function* relationship between the malfunctions[5]. Continuing with the example system in Figure 1, the malfunction hierarchy for the reactor system is shown in Figure 3. Each node in the malfunction hierarchy represents the hypothesis that some particular malfunction has occurred. Note that the nodes located in the upper levels of the hierarchy (Pressure System Failure, Cooling System Failure) represent more abstract malfunction hypotheses then those lower in the hierarchy (Relief Valve Failure, Condenser Failure). Further note that the sub nodes of any node are more particular kinds of the super node. For example, a Relief Value Failure is a particular kind of Pressure System Failure. Therefore, as one traverses the hierarchy in a top down fashion, one examines more detailed hypotheses about what malfunction has occurred in the system.

[4]Space limits this paper to the compiled system issues, see reference [9] for a detailed discussion of the deep model issues and computational strategies.

[5]Though the simple examples of this paper use only a single hierarchy, other work [10] recognizes that multiple hierarchies may be required to properly represent all system malfunctions.

Each node in the hierarchy has knowledge about the conditions under which the malfunction hypothesis it represents is plausible. Each node of the malfunction hierarchy is therefore a small expert system that evaluates whether the malfunction hypothesis it represents is present given the data. While there are a number of ways this could be accomplished, conceptually what is required is pattern-matching based on data *features*. Each node contains a set of features that are compared against the data. The results of this comparison indicate the likelihood of that particular malfunction being present. The pattern matching structure of a node in the CSRL language [1] is called a *knowledge group*. The knowledge groups compare relevant features against the data and yield a symbolic likelihood.

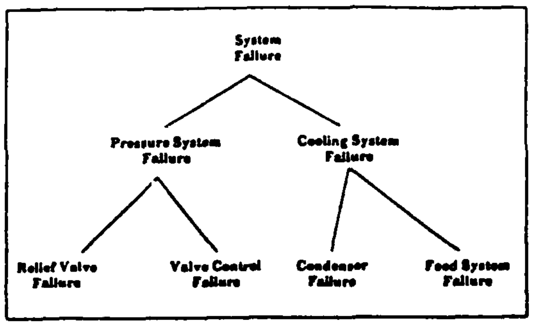
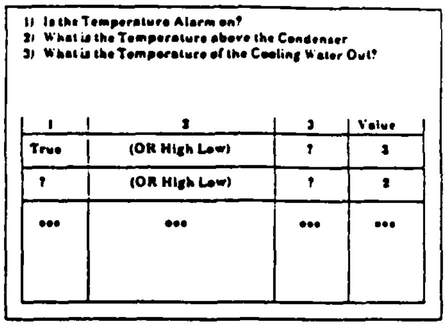


Figure 3: Hierarchy of malfunctions from Figure 1

Consider the knowledge group depicted in Figure 4 taken from the Cooling System Failure node of our example. The first section represents three queries about a datum value[6]. Each column of the table underneath represents a possible answer to each question (column 1 to question 1, etc.). The match value assigned to the knowledge group is based on the value located at the end of each row[7]. In our example when the answer to question 1 is True, the answer to question 2 is either High or Low and regardless of the answer to question 3, row 1 assigns a value of 3 to the knowledge group. The rows of the table are evaluated in order until either a row of queries matches or no row matches and a default value is assigned. Thus, when the data pattern of row 1 exists, the knowledge group (and thus the malfunction) is established at a high level of confidence.

Finally, the control strategy of a hierarchical classifier is termed *establish-refine*. In this strategy, each node is asked to establish how likely the malfunction hypothesis it represents is given the data. The node, using knowledge groups, determines an overall measure of likelihood. If the node establishes, i.e., the malfunction is judged likely, then each sub of that node are asked to try and establish themselves. If a node is found to be unlikely, then that node is ruled-out and none its subs are evaluated.

Consider the example using the data from Figure 2 and the hierarchy of Figure 3. The top node is established since the temperature is high. Each of the subnodes is then asked to establish. Cooling System Failure rules out and none of its subnodes are examined. Pressure Relief Failure establishes and its subs are asked to try and establish themselves. This process



1) Is the Temperature Alarm on?
2) What is the Temperature above the Condenser
3) What is the Temperature of the Cooling Water Out?

1	2	3	Value
True	(OR High Low)	?	3
?	(OR High Low)	?	2
•••	•••	•••	•••

Figure 4: A Knowledge Group from Cooling System Failure

continues until there are no more nodes to examine. In this way, the most specific malfunctions that can be confirmed are given as the diagnostic conclusion[8].

3. Data Validation

Two important methods are available for validating data in conjunction with hierarchical classification. First, it is possible to establish a malfunction based on a preponderance of other evidence. If the node can establish but not all the data it expects is present, the data not meeting expectation is subject to question. In the original example, the Pressure Relief System Failure established despite a normal pressure reading based on a preponderance or other evidence.

Secondly, intermediate diagnostic conclusions from other nodes provide a context to evaluate data. If the Pressure System Failure does establish, its subs can expect the pressure reading to be abnormal. If it is not, they can also question the pressure reading.

In the remainder of this section, we will discuss the following aspects of a data validation system:

1. How data is questioned based on diagnostic expectations.
2. The various methodologies available that could resolve the questionable data.
3. How the normal control flow of diagnostic problem solving is affected.

3.1. Questioning a Datum Value

Discovering a questionable datum involves two steps. First, set some expectations, using local knowledge or the context of other nodes. Second, use those expectations to flag some particular data value as questionable.

The expectations of a malfunction are embodied in the knowledge group. The knowledge group mechanism was designed to give a rating of pattern fit to data. If the fit is not as expected, those data values not meeting expectations are identified as questionable. In the example of Pressure Relief Valve Failure, evidence exists that the valve has failed even though the pressure is normal. The lack of fit between data and pattern allow the pressure value to be identified as questionable. Diagnosis continues despite apparent data conflict since enough evidence exists for establishing the malfunction hypothesis.

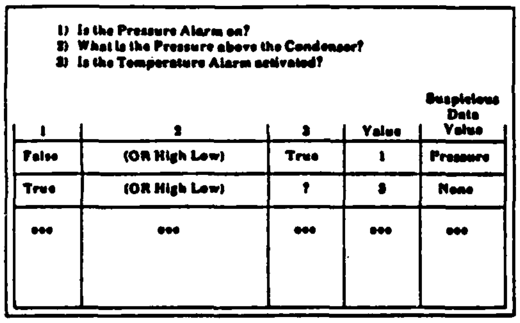
[6]In the present CSRL implementation, these values are fetched from a database, though other means may be used, such as calls to deep models, simulations, etc.

[7]In this case, the values assigned are on a discrete scale from -3 to 3, -3 representing ruled-out and 3 representing confirmed.

[8]Note that if multiple conclusions are reached, then either multiple *independent* malfunctions have occurred or multiple *dependent* malfunctions must be resolved into a smaller set [8].

Furthermore, the expectations of previous nodes create a context of expectation for the node currently being examined. Consider the example hierarchy of Figure 3. In order to establish the malfunction hypothesis Relief Value Failure, the malfunction hypothesis Pressure System Failure must have established. In the context of considering the Valve Failure, some expectations were created based on the establishment of the Pressure System Failure node and other ancestors. Since those expectations always exist when considering Valve Failure, i.e., you can't get to Valve Failure without establishing Pressure System Failure, they can be coded into the Valve Failure Node.

How expectations are used for the Pressure Relief System Failure node is shown in Figure 5. A modification is made to the standard knowledge group of Figure 4 that allows the expert to indicate both a match value for the group and a set of data that do, not meet the expectations established at this stage of the problem solving. Thus, Pressure Relief Failure establishes (based on other data features) despite the lack of a change of pressure. However, in establishing the node, one should question why the pressure did not change. This is done by placing the pressure value in the rightmost column of the matching row. If that row is matched, then the match value is returned but the indicated data value is placed in a list of questionable data values which will be examined later. If the match value is high enough, the node establishes despite the existence of conflicting data. That is, if there is enough evidence to show a malfunction despite a conflicting value, the problem solving may continue. However, it may be the case that the value being questioned is of vital importance to establishing the node. The match value will reflect this, the node will not establish, but the data will still be placed on the questionable data list. After an initial run of the problem solver, the questionable data list will consist of data values that did not meet the expectations of some node.


1) Is the Pressure Alarm on?
2) What is the Pressure above the Condenser?
3) Is the Temperature Alarm activated?

1	2	3	Value	Suspicious Data Value
False	(OR High Low)	True	1	Pressure
True	(OR High Low)	?	3	None
•••	•••	•••	•••	•••

Figure 5: Knowledge Group Modified for Data Validation

3.2. Questions Regarding Combinatorial Explosion

The knowledge engineer is responsible for providing the node with both feature matching data and datum values that do not meet expectations of the malfunction hypothesis at that point. This, as mentioned previously, is a compiled approach to data validation. It may appear that such a compilation of possibilities will result in a combinatorial explosion. However, not all possible *combinations of sensor readings need be encoded; only those* situations which the expert deems reasonable or necessary need be placed in the knowledge groups. More importantly, in our approach to use of knowledge groups, a hierarchy of abstractions is used to go from the data to the classificatory conclusion [3]. Thus, *the set of data elements needed for any node* in the hierarchy is limited to only those relevant for the malfunction hypothesis it represents. Furthermore, the data elements of each node are subdivided among the knowledge groups that need them. That is, even within the node, the data is further partitioned to only those knowledge groups that will use that data. The knowledge engineer is therefore presented with a much simpler task. Only those combinations of a few data items that present a missed expectation need be encoded into the diagnostic system.

3.3. Resolving the Values of Questionable Data

A number of methods are available for resolution of questionable data. The system may examine the state of the nodes in the diagnostic hierarchy to see if any other nodes were satisfied or unsatisfied with the datum value in question. If many other nodes also questioned the same datum, increased evidence exists that the value is incorrect. In contrast, if no other node who used it questioned the value, evidence for its incorrectness is decreased and the validity of the knowledge group that originally questioned the value is suspect. This is a *redundancy of usage* by *the nodes in the diagnostic hierarchy.*

If a node indicates a data conflict that prevents it from establishing, that node can examine its subnodes to see if they have any expectations that resolve the conflict. Using Figure 3 as an example, assume that Cooling System Failure cannot establish because it requires that a temperature datum be high. The system could examine the subnodes of Cooling System Failure, namely Feed System Failure or Condenser Failure, to see if they can establish independent of the temperature reading. If Feed System Failure has enough data evidence to establish anyway, then its expectations might resolve the issue of the temperature reading. If it too expected a high temperature that does not exist, there is increased evidence that the temperature reading is incorrect. If it did not have any expectations about the temperature reading, then the knowledge found in the Cooling System Failure could be questionable. This is a *redundancy of expectation.*

The diagnostic system could contain more involved or detailed hardware checks then would be feasible to perform on all sensors as the data becomes available. General methods are invoked for a particular kind of sensor (thermocouples vs. mechanical thermal sensors) that are too computationally intensive/expensive to run all the time on incoming data. These procedures are run if there is evidence that their use would pay off. Such evidence is provided by a data conflict discovered by the diagnostic system. While these methods do not depend directly on information provided by the problem solving state (as the above methods do), they are only used when indicated by the diagnostic system. This is a *hardware redundancy of need.*

3.4. Control Issues

While sections 3.1 and 3.3 have discussed discovering and resolving possibly invalid data, this section addresses the issues of control flow changes to the normal establish – refine strategy.

1. What happens to data whose values have been proven to be either incorrect or unresolved? If found to be incorrect, the value in the data base must be modified, i.e., the central data base value modified, to indicate the change. If unresolved, it must be flagged as such in hopes of being resolved later.

2. If incorrect data has been found, then it is possible that the problem solver made some mistakes in its diagnosis. It may be necessary to re-run the hierarchy to see if the diagnostic conclusions change. Furthermore, any unresolved data may be resolved as a result of the new information.

3. The basic control strategy would then look like the following cycle:

a. Run the Hierarchy, finding questionable values.

b. Run resolution techniques of section 3.3 to resolve the values if possible.

c. Update the data base with any changes.

This cycle continues until either no data is questioned or the user sees an answer that is satisfactory.

4. Other control strategies are also available. The resolution techniques of section 3.3 could be run as soon as a data item is questioned, i.e., right in the middle of the problem solving. This requires a backtracking scheme that re–runs any other node that used that value. Finally, the operator can be directly involved in changing data values at any step of the process based on his/her expert opinion of the situation.

4. Implementation to Date

The majority of the structures and strategies indicated in the paper have been included into a version of the CSRL [1] tool as it has been applied to diagnosis of Cooling System Accidents in a Boiling Water Nuclear Power Plant. The knowledge groups have been modified as indicated in Figure 5 such that a datum value is questioned whenever it does not meet expectation. A number of scenarios of data conflict have been encoded into this system concerning the plant and more are being added. The control strategies that allow the hierarchy to be re–run until either no data is questioned or the user is satisfied have been implemented. Future work will concentrate on problems of backtracking to prevent re–running the entire hierarchy. With regards to data value resolution, present work is focusing on methodologies for resolving questionable sensors as indicated in Section 3.3. To date, each kind of sensor has associated with it a set of hardware checks that are not normally invoked. Furthermore, if the additional hardware checks do not resolve the problem, the user is presented with the conflict and information on all the nodes that used that value, including whether that value was questioned there or not.

5. Conclusion

While some data validation can be done using the standard techniques of hardware redundancy, a higher level of redundancy based on *diagnostic expectations* can address more of the issues of conflicting data in a more sensible manner. Intermediate diagnostic conclusions from hierarchical diagnost[illegible]ons provide expectations that can indicate invalid data values. Note that the method is deceptively simple. This is due to the *focus* provided by chunking the match and expectation knowledge into the smaller, more manageable parts in the knowledge groups. Furthermore, the hierarchy also acts to simplify the complexity of the match and expectation knowledge due to the context of information provided by the establishment or rejection of parent nodes. Despite this, much power can be gained in both diagnosis and sensor/data validation by use of these simple methods. Some advantages include:

1. Questioning data based on diagnostic expectations provides a way to focus on only some data, as opposed to hardware methods which must worry about all data *a priori*.

2. Even if a data value is questioned, the diagnostic process can continue if other evidence exists for the failure. Thus the system can work with conflicting data.

3. Such a system integrates well with existing systems that presently rely solely on hardware redundancy by using those values as data for both diagnosis and a higher level of data validation.

4. The programming by a user of such a system is facilitated by existing tools (CSRL) that need only minor modifications.

Acknowledgments

We would like to thank Don Miller, Brian Hajek and Sia Hashemi of the Division of Nuclear Engineering at the Ohio State University for their help in developing and implementing these ideas. Also, one of the authors (Punch) would like to thank Mike Tanner for his invaluable aid in criticizing this paper and sharpening its ideas.

References

1. T. C. Bylander / S. Mittal. CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling. *AI Magazine* 7(Summer 1986).

2. B. Chandrasekaran / S. Mittal. Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems. In *Advances in Computers*, M. Yovits, Ed., Academic Press, 1983, pp. 217–293.

3. B. Chandrasekaran. From Numbers to Symbols to Knowledge Structures: Pattern Recognition and Artificial Intelligence Perspectives on the Classification Task. In *Pattern Recognition in Practice – II*, North Holland Publishing, 1986, pp. 547–559.

4. W. J. Clancey. Heuristic Classification. *Artificial Intelligence 27*, 3 (1985), 289–350.

5. J. F. Davis / W. F. Punch III / S. K. Shum / B. Chandrasekaran. Application of Knowledge – Base Systems for the Diagnosis of Operating Problems. Presented to AIChe. Annual Meeting, Chicago Ill. 1985.

6. Gomez, F. / Chandrasekaran, B. Knowledge Organization and Distribution for Medical Diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics SMC – 11*, 1 (January 1981), 34–42.

7. S. Hashemi, B.K. Hajek, D.W. Miller, B. Chandrasekaran, and J.R. Josephson. Expert Systems Application to Plant Diagnosis and Sensor Data Validation. *Proc. of the Sixth Power Plant Dynamics, Control and Testing Symposium*, Knoxville, Tennessee, April, 1986.

8. J. R. Josephson, B. Chandrasekaran and J. W. Smith, M.D. Abduction by Classification, Assembly, and Criticism. Revision of an earlier version called "Abduction by Classification and Assembly" which appears in *Proc. Philosophy of Science Association, Vol.1, Biennial Meeting, 1986.* 1986.

9. V. Sembugamoorthy / B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In *Experience, Memory and Reasoning*, J. L. Kolodner and C. K. Riesbeck, Eds., Erlbaum, 1986, pp. 47–73.

10. J. Sticklen. MDX2: An Integrated Diagnostic Approach. Dissertation In Progress, Ohio State University. 1987.

Integrating Model-Based Reasoning and Case-Based Reasoning for Design Problem Solving

Ashok Goel
B. Chandrasekaran

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University

August 10, 1988

This paper is to appear in the *Proceedings of the AAAI Workshop on AI in Design*, St. Paul, Minnesota, August 1988.

Integrating Model-based Reasoning and Case-based Reasoning for Design Problem Solving

Ashok Goel and B. Chandrasekaran

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University

Abstract

Design is a complex information processing activity. What problem solving strategy is appropriate for the design of a specific artifact depends on what knowledge is available to the designer. One form in which design knowledge is often available is that of previously designed artifacts. For this reason, case-based reasoning is an attractive approach to design problem solving. A main issue in case-based design is how to adapt the structure of an existing design to achieve a new device functionality. This capability requires a causal understanding of how the structure of a device enables the accomplishment of its function. Such a causal understanding can often be expressed as a functional model of the device that specifies the designer's knowledge about the role of the various components and their relations in the functioning of the device. The Functional Representation scheme is a method for organizing and representing an agent's causal understanding of devices. In this scheme, the agent's understanding of the functioning of a device is expressed as behaviours that compose the functions of its structural components into device functions. In this paper, we illustrate how this organization of knowledge may enable a designer to identify the portions of the device structure that need to be modified to achieve a new functionality, and to reason about the effects of these structural changes. The integration of this capability with that of indexing, storing, and retrieving previous design cases can provide a powerful strategy for efficiently solving complex design problems.

1. Design Problem Solving

Design in general is a very complex information processing activity. Moreover, design covers a very wide variety of phenomena: planning a day's work, composing a computer program, and constructing a scientific theory can all be viewed as instances of design problem solving. In order to provide some focus to the present discussion, let us restrict its scope to the design of engineering artifacts that accomplish some explicitly stated goals.

The information processing task of design can be abstractly characterized as a specification of a set of components and their relations that meet a set of constraints [4]. Some of the input constraints may represent the desired functions of the artifact, some may pertain to the components and their relations, while others may refer to the design process itself. The components and their relations specified in the output comprise the structure of the artifact that enables the accomplishment of the intended functions. This structure is built out of primitive components (*e.g.* a resistor) and primitive relations (*e.g.* serial connection) available to the designer as part of his domain knowledge. Since the number of primitive components and relations in a given domain can be very large, synthesizing a structure that enables the achievement of a given function can often be open-ended and computationally intractable.

Perhaps because design is so challenging, it has attracted substantial attention in Artificial Intelligence research. A number of mechanisms for design problem solving have been proposed in the literature: *e.g.*, plan synthesis using abstract objects and operators [10], production rule architectures for configuration problems [9], relaxation techniques for design by multiple constraint satisfaction [8], etc. Although each of these techniques has relative advantages and drawbacks, none of them is right or wrong *per se*: for simple design problems, almost any of them is likely to work; for complex problems, none of them by itself may succeed.

An important issue in design problem solving is how well the form in which knowledge is available to the designer matches the form in which knowledge is needed to perform a given design task computationally efficiently. For instance, if for a specific routine design task, the designer knows of precompiled skeletal design plans which enable him to recursively decompose the design task into subtasks upto the level of choosing primitive components, then the design task can be efficiently performed by plan instantiation and refinement [1,5].

2. Case-based Design

One form in which design knowledge is often (typically?) available to a designer is that of previously designed artifacts. If the designer is charged with the task of designing a device that accomplishes a specific function and has previously come across a device that achieved a similar function, then he may design the new device by suitably adapting the known design of the existing device [14].

In the canonical example of this approach [7,11], the designs of artifacts known to the designer are stored in memory as design *cases*. A design case may be indexed by using complex multiple indices that may specify the functions of the artifact and the prominent features of its design. When the designer is supplied with the specifications of a new device, the design case that best matches the specifications is retrieved, and modified to meet the desired specifications. The generated design can then be tested, for instance, by *qualitative simulation*. If the test is unsuccessful, then the design can be further modified; and if it is successful, then it is stored as another design case. In this way, the designer may *acquire* new design knowledge. Moreover, when storing a new design case in memory, the designer may *generalize across index categories* leading to further *operationalization* of his design knowledge.

Case-based reasoning is an attractive approach to design problem solving because it makes use of experiential knowledge, and can provide mechanisms for acquisition and operationalization of design knowledge. In general, case-based design is likely to be computationally more efficient than approaches that perform design from "first principles" for every new problem.

A main issue in the case-based approach to design is how to adapt the structure of an existing design to achieve a new functionality. This capability requires a *causal understanding* of how the structure of a device enables the accomplishment of its function. Such a causal understanding can often be expressed as a *functional model* of the device that specifies the designer's *dynamic knowledge* about the role of the various components and their relations in the functioning of the device. It is this causal understanding that enables a designer to identify the portions of the device structure that need to be modified to accomplish a new function, and to reason about the effects of these structural changes on the device functionality.

3. Functional Modeling of Devices

Over the last several years our research group has been developing a Functional Representation *scheme* to capture an agent's causal understanding of devices [2,3,6,12]. A central thesis of this scheme is that problem solving agents often understand the functioning of a complex device by decomposing the device function into the functions of its structural components. The functioning of a component is similarly understood in terms of the functions of its subcomponents. This decomposition may go on upto as many levels as needed, with only limited interactions between a few components at any level. In the recomposition phase, the functions of the components are composed by behaviors to obtain the function of the device. The function of a device component is similarly obtained by behaviors that compose the functions of its subcomponents. The specification of a behavior at any level may include pointers to deeper knowledge and assumptions underlying the recomposition at that level.

4. Functional Representation of Feedback

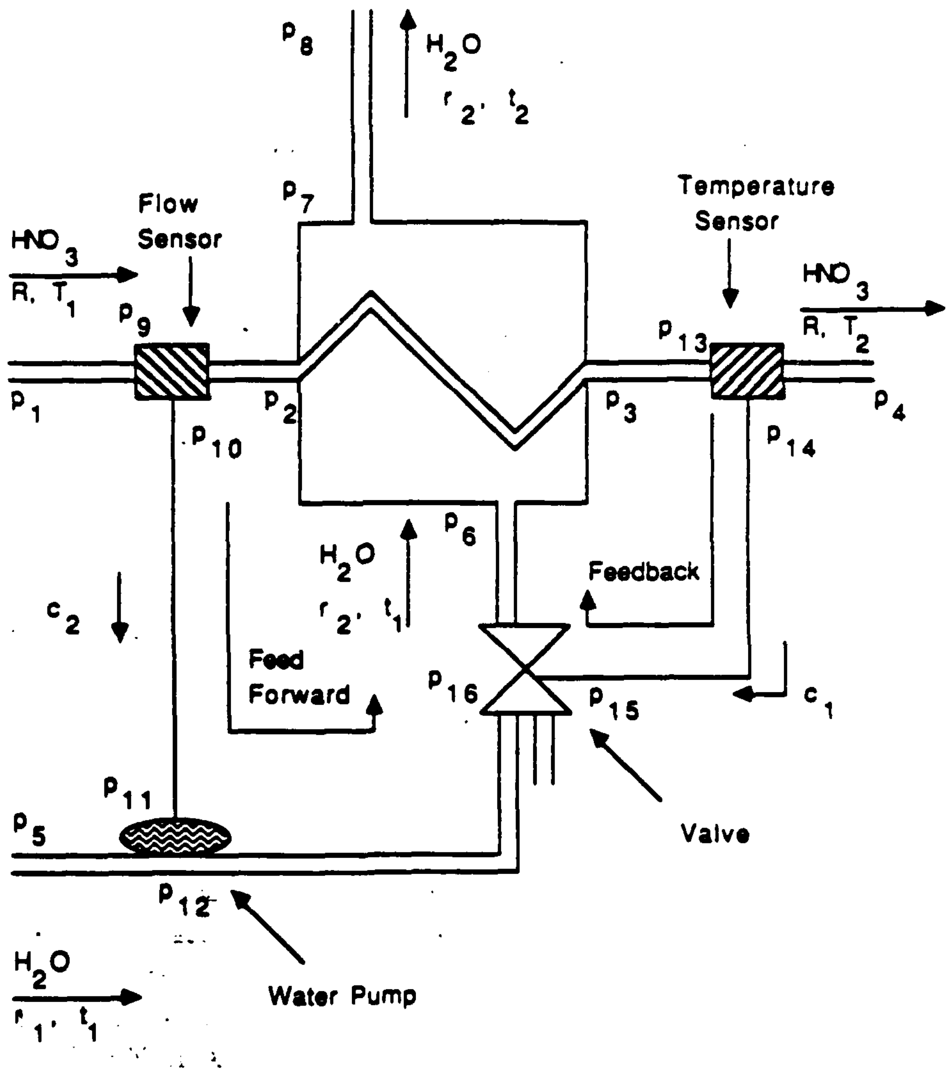


Figure 1: The Nitric Acid Cooler

Let us illustrate the Functional Representation scheme by considering an agent's understanding of the functioning of the the Nitric Acid Cooler (NAC) shown schematically in Figure 1 [6]. Hot Nitric Acid (HNO_3) enters the cooler at p_1 with flow rate R and temperature T_1, and exits at p_4 with the same flow rate and a lower temperature T_2, where p_1, p_2 ... are points in the device space. Similarly, cold water (H_2O) is pumped into the cooler at p_5 with flow rate r_1 and temperature t_1, and exits at p_8 with flow rate r_2 and a higher temperature t_2. Inside the heat exchange chamber, heat is transferred from hot Nitric Acid to cold water, thereby cooling Nitric Acid from T_1 to T_2 and heating water from t_1 to t_2.

The flow rate R of the inflowing Nitric Acid is measured by a flow sensor, and information about perturbations in its value is communicated to the water pump by a signal c_1 in the wire connecting the sensor and the pump. The pump regulates the rate r_1 at which water flows into the cooler to reflect the perturbations in value of R. This is an example of *feedforward* control. Similarly, the temperature T_2 of outflowing Nitric Acid is measured by a temperature sensor, and information about perturbations in its value is communicated to the valve by a signal c_2 in the wire connecting the sensor and the valve. The valve regulates the rate r_2 at which water enters the heat exchange chamber to reflect the perturbations in the value of T_2, and releases excess water from the device. This is an example of *feedback* control.

4.1. Function

The functional representation of a problem solving agent's causal understanding of NAC consists of three parts. In the first, the functions of NAC are expressed as hierarchically organized schemata, in which the nodes are the intrinsic functions of the device and its components, and the arcs are the relations between these functions. Some of the higher level functions of NAC and the relations between them are shown in Figure 2. We note that this part of the understanding of NAC is in terms of functional abstractions independent of the specific device structure on which the functions are realized. Thus, CoolNitricAcidToT_2 is the primary function of NAC. HeatWater is the secondary function of the device; it is also a side function of CoolNitricAcidToT_2.

At the next level in the network of NAC functions, SupplyWaterToChamberAtRater_2 is a subfunction (or constituent function) of HeatWater; it is also a supporting function for CoolNitricAcidToT_2, *i.e.* its function is to satisfy the preconditions for the accomplishment of CoolNitricAcidToT_2. Similarly, SupplyNitricAcidToPipeInChamber is a subfunction of CoolNitricAcid and a supporting function of HeatWater. This captures an agent's understanding of the *interaction* between the functions of CoolNitricAcidToT_2 and HeatWater, allowing him to reason that since the subfunction for CoolNitricAcidToT_2 is a supporting function of HeatWater and *vice versa*, the Nitric Acid will get cooled if, and only if, water simultaneously gets heated. Further, this enables the agent to view the role of functions from *multiple*

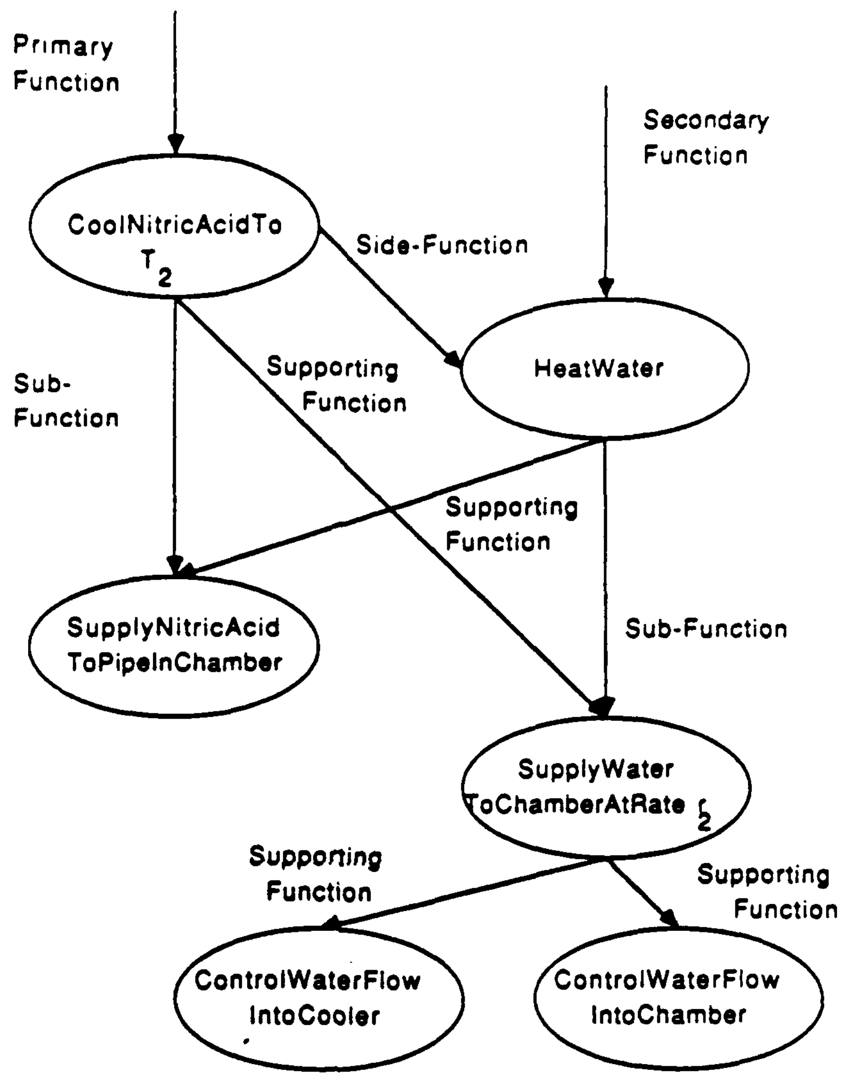


Figure 2: Functional Organization of the Nitric Acid Cooler

perspectives: SupplyWaterToChamberAtRater$_2$ is a subfunction from the perspective of achieving HeatWater, but a supporting function from the perspective of accomplishing CoolNitricAcidToT_2. At the next lower level in the network of NAC functions, the feedback and feedforward functions of ControlWaterFlowIntoChamber and ControlWaterFlowIntoCooler are similarly understood as supporting functions for SupplyWaterToChamberAtRater$_2$.

The schemas for some of these functions are shown in Figure 3. The underlined expressions in the figure are the primitives of a Functional Representation Language, each with an associated semantics. The primitives Given and ToMake provide an input-output specification of the functions, while By specifies the behavior that results in the

Function: CoolNitricAcidToT_2
Given:
HNO_3 at p_1 with Flow Rate R and Temperature T_1
ToMake:
HNO_3 at p_4 with Flow Rate R and Temperature T_2
By: Behavior1
Provided:
H_2O at p_6 with Flow Rate r_2 and Temperature t_1
End Function CoolNitricAcidToT_2

Function: HeatWater
Given: H_2O at p_5 with Temperature t_1
ToMake:
H_2O at p_8 with Flow Rate r_2 and at Temperature t_2
By: Behavior2
Provided:
HNO_3 at p_2 with Flow Rate R and Temperature T_1
End Function HeatWater

Function: SupplyWaterToChamberAtRater_2
Given: H_2O at p_5 at Temperature t_1
ToMake: H_2O at p_7 with Flow Rate r_2 and Temperature t_1
By: Behavior4
Provided:
(i) Control Signal c_1 at p_{11}
(ii) Control Signal c_2 at p_{15}
End Function SupplyWaterToChamberAtRater_2

Function: ControlWaterFlowIntoChamber
Given: HNO_3 at p_{13} with Flow Rate R and T
ToMake: Control Signal c_2 at p_{15}
By: Behavior6
End Function ControlWaterFlowIntoChamber

Figure 3: Schemas for Some Functions of NAC

accomplishment of the function. Thus each function in the network can be used to *index* the behaviors responsible for accomplishing it. Provided specifies the states of the device in which only a given function can be accomplished, and relates the function to its supporting functions.

4.2. Structure

In the second part of an agent's causal understanding of NAC, the structure of NAC which realizes the above functions is expressed in terms of the primitive components and their relations. The specification of the structure also contains schemas for the functional abstractions of the components. This is shown in Figure 4 and 5. Thus, Chamber

```
Begin Components
  FlowRateSensor {$p_9, p_{10}, p_2$}
  WaterPump {$p_5, p_{11}, p_{12}$}
  TemperatureSensor {$p_3, p_{13}, p_{14}$}
  Valve {$p_{15}, p_{16}, p_6$}
  Chamber {$p_2, p_6, p_3, p_7, space_1$}
  Pipes {$p_1, p_2$},
      {$p_2, p_3, space_2$},
      {$p_3, p_4$},
      {$p_5, p_{16}$},
      {$p_{16}, p_6$},
      {$p_7, p_8$}
  Wires {$p_{10}, p_{11}$},
      {$p_{14}, p_{15}$}
End Components

Begin Abstractions-of-Components
  Component: FlowRateSensor {$p_9, p_{10}, p_2$}
    Functions: Measure Flow Rate of Fluids

  Component: WaterPump {$p_5, p_{11}, p_{12}$}
    Functions: Pump Water

  Component: TemperatureSensor {$p_3, p_{13}, p_{14}$}
    Functions: Measure Temperature of Fluids

  Component: Valve {$p_{16}, p_{15}, p_6$}
    Functions: Regulate Flow Rate of Fluid

  Component: Chamber {$p_2, p_6, p_3, p_7, space_1$}
    Functions: Contain Fluid, Transport Fluid
End Abstractions-of-Components
```

Figure 4: Partial Specification of the Structure of NAC: Some Components and their Abstractions

{p_2, p_6, p_3, p_7, *space*$_1$} is a component of NAC, the *space*$_1$ enclosed by the chamber includes the *space*$_2$ enclosed by Pipe {p_2, p_3, *space*$_2$}, and the functions of the chamber are to contain fluid and transport fluid. We will not devote much space here to the issue of representation of structure except to note that the functional abstractions of the components are device independent.

4.3. Behavior

In the third part of an agent's causal understanding of NAC, the behaviors that compose the functional abstractions of the structural components into the device functions are represented as acyclic directed graphs in which the vertices are partial states of the device and the edges are causal state transitions. The directed graphs for the behaviors that achieve some of the NAC functions discussed earlier are shown in Figures

Begin **<u>Relations</u>**
  <u>Serially</u> <u>Connected</u>:
    Pipe $\{p_1, p_2\}$,
    Pipe $\{p_2, p_3, space\text{-}(2)\}$,
    Pipe $\{p_3, p_4\}$

  <u>Serially</u> <u>Connected</u>:
    Pipe $\{p_5, p_{16}\}$,
    Valve $\{p_{16}, p_{15}, p_6\}$,
    Pipe $\{p_{16}, p_6\}$,
    Chamber $\{p_2, p_6, p_3, p_7, space_1\}$,
    Pipe $\{p_7, p_8\}$

  <u>Serially</u> <u>Connected</u>:
    FlowRateSensor $\{p_9, p_{10}, p_2\}$,
    Wire $\{p_{10}, p_{11}\}$,
    WaterPump $\{p_5, p_{11}, p_{12}\}$

  <u>Serially</u> <u>Connected</u>:
    TemperatureSensor $\{p_3, p_{13}, p_{14}\}$,
    Wire $\{p_{14}, p_{15}\}$,
    Valve $\{p_{16}, p_{15}, p_6\}$

  <u>Includes</u>: $\{space_1, space_2\}$
End **<u>Relations</u>**

Figure 5: Partial Specification of the Structure of NAC: Relations between some Components

6 and 7. The primitive <u>Using-Function</u> specifies the function of some component that is used by the behavior in accomplishing some higher level function, while <u>By</u> refers to some lower level behavior.

The specification of a behavior may include pointers to deeper causal knowledge and assumptions underlying a causal state transition in the behavior. For instance, Behavior1 for accomplishing the function of CoolNitricAcidToT_2 uses (Generic-Knowledge1) that may be stated as follows: In accordance with the Zeroth Law of Thermodynamics, in the context of the Chamber$\{p_2, p_6, p_3, p_7, space_1\}$ enclosing the Pipe$\{p_2, p_3, space_1\}$, heat will flow from hot Nitric Acid to cold water resulting in a decrease in the temperature of Nitric Acid from T_1 to some T_2 and an increase in the temperature of water from t_1 to some t_2.

Similarly, Behavior1 accomplishes CoolNitricAcidToT_2 under Assumption1 which may be stated as follows: The relation between temperature T_1 and flow rate R of inflowing Nitric acid, the desired temperature T_2 of outflowing Nitric acid, and the temperature t_1 and flow rate r_2 of water flowing into the heat exchange chamber, is such

Behavior1
ToAchieveFunction: CoolNitricAcidToT_2
HNO_3 at p_1
with Flow Rate R and Temperature T_1

↓ **By: Behavior3**

HNO_3 at p_2
with Flow Rate R and Temperature T_1

↓ **Under-Condition: H_2O at p_7 with Flow Rate r_2 and Temperature t_1**
As-Per: Generic-Knowledge1
With: Assumption1
Using-Function: Transport Fluid of Pipe $\{p_2,p_3\}$

HNO_3 at p_3
with Flow Rate R and Temperature T_2

↓ **Using-Function: Transport Fluid of Pipe $\{p_3,p_4\}$**

HNO_3 at p_4
with Flow Rate R and Temperature T_2
End Behavior1

Figure 6: Behavior1 of NAC

that the capacity of water to absorb heat in the chamber exceeds the capacity of Nitric Acid to release heat.

The interactions between the functions of a device are, of course, reflected in the behaviors that accomplish the functions. For instance, Behavior1 for accomplishing the function of CoolNitricAcidToT_2, and Behavior2 for achieving HeatWater shown in Figures 4(a) and 4(b) respectively, interact in that Behavior1 will result in cooling Nitric Acid to T_2 if and only if Behavior2 simultaneously results in heating water. This interaction is being captured by the primitive Under-Condition which specifies that the causal transition from one device state to another in some behavior is conditional on some other device state being true.

Thus, the behaviors for accomplishing these interacting device functions are *nonlinear* in the same sense that the plans to achieve interacting goals are often nonlinear

Behavior2

ToAchieveFunction: HeatWater

H_2O at p_5 at Temperature t_1

$\Big\downarrow$ By: Behavior4

H_2O at p_7
with Flow Rate r_2 and Temperature t_1

$\Big\downarrow$ Under-Condition: HNO_3 at p_2 with Flow Rate R and at Temperature T_1

As-Per: Generic-Knowledge1

Using-Function: Transport Fluid of Chamber$\{p_7, p_3, p_8, p_2\}$

H_2O at p_7
with Flow Rate r and Temperature t_2

$\Big\downarrow$ Using-Function: Transport Fluid of Pipe $\{p_7, p_8\}$

H_2O at p_8
with Flow Rate r and Temperature t_2

End Behavior2

Figure 7: Behavior2 of NAC

[10]. That is, while the device behaviors can be *partially ordered*, each individual behavior being a linear sequence of causal state transitions, a total ordering of the behaviors is not possible. Instead, a network of behaviors mirroring the network of Figure 2 collectively results in the functioning of the device. In fact, for the specific case of the NAC, the device behaviors are inherently non-serializable. Thus, if a problem solving agent were to perform a qualitative simulation to verify whether Behavior1 will indeed lead to cooling of Nitric Acid to T_2, then he will have to perform "in parallel" a simulation to check if Behavior2 indeed results in heating water. Since the causal state transitions are already compiled, *qualitative simulation* in the Functional Representation scheme is performed not from "first principles" or by using naive physics, but by tracing the sequences of behavioral states organized around the device functions.

5. Model-based Design

5.1. Design Example 1

Let us illustrate how the functional representation of a designer's causal understanding of a device in conjunction with case-based reasoning is useful for design problem solving. Let us assume that the designer has stored in his memory the design of NAC, along with its functional representation, indexed by its primary and secondary functions. Let us suppose that the designer is charged with the design of a Sulphuric Acid Cooler (SAC), and retrieves from his memory the design and functional representation of NAC.

Now, if the designer has access to a knowledge-base of general substances and their properties, then he may determine that both HNO_3 and H_2SO_4 are acids. He may perform a qualitative simulation to check if the design of NAC will also work for cooling Sulphuric Acid by tracing the compiled sequences of behavioral states indexed by the device functions. If the simulation succeeds (let us assume that it does), then the designer has a design for SAC that can be stored in memory. The designer may generalize across the case index categories of Nitric Acid Cooler and Sulphuric Acid Cooler, and learn the index category of the general Acid Cooler. He may organize his memory in a classificatory hierarchy of three nodes: the root node of a general Acid Cooler, and leaf nodes of its two specific instances known to the designer. Moreover, the functional representation of NAC can also be generalized and stored at the root node of the classificatory hierarchy. We note also that the functional representation provides a causal *explanation* for the newly designed SAC.

5.2. Design Example 2

Let us consider a second example in which the designer is charged with the design of a Sulphuric Acid Cooler that cools H_2SO_4 by as much as, say, *100* degrees Celsius. Let us suppose that the designer retrieves from his memory the design for the old SAC that can cool H_2SO_4 by only, say, *70* degree Celsius. This is an instance of the more general design problem in which the operating range of the new device is different from that of the old one.

Again, the designer may begin by performing a qualitative simulation on the functional representation of the old SAC. This time, however, lets suppose that the simulation fails because Assumption1 is violated (the capacity of water in the Chamber $\{p_2, p_6, p_3, p_7, space_1\}$ to absorb heat exceeds the capacity of Nitric Acid in Pipe $\{p_2, p_3, space_2\}$ to release heat). The important point here that the functional representation scheme allows the designer to directly detect the violation of this assumption by merely tracing compiled sequences of behavioral states in Behavior1 and checking the Assumption1 along the way.

Since the flow rate and temperature of water flowing into the chamber are the only variables in the statement of Assumption1, the designer may decide to increase the flow rate of water. He may determine from the functional abstractions of the components and by tracing Behaviors 2 and 4 that changing the WaterPump or the Valve may help. He may select a WaterPump of a larger capacity from his knowledge base of available primitive components, and *formulate a constraint* that specifies that the components downstream to WaterPump in any behavioral sequence must be able to support the increased flow rate of water. When he now performs a qualitative simulation, he may *propagate* this constraint from one behavioral state to another. In this way, he may find that while Assumption1 is now being satisfied, the amount of water being pumped exceeds the capacity of the Valve. Again, he may select a different Valve from his knowledge base of primitive components that supports the increased flow rate of water and thereby *satisfies* the constraint. This method of reasoning about the effect of structural changes on devices is close to that of *constraint posting* [13]. At end of this process, he has a design for the new SAC that can be stored in memory.

5.3. Design Example 3

Let us consider as a third example, the design of SAC that cools Sulphuric Acid by as much as, say, *500* degrees Celsius. Let us suppose the designer follows the same procedure as in the second example, but cannot find any way to satisfy Assumption1. However, if the designer knows of *cascading as a generic mechanism*, then he may design a new SAC by composing an array of old SACs. This requires a knowledge base of generic mechanisms such as cascading, feedback, etc. Each such mechanism may be stored in the form of a device-independent *functional template*. In fact, the Functional Representation scheme itself can be viewed as a template for organizing the designer's knowledge of devices and mechanisms.

The cascading mechanism can be functionally represented as a decomposition of the device function into several identical unspecified subfunctions, each of which is accomplished by the same unspecified behavior. These subfunctions are recomposed into the device function by another higher level behavior. In other words, the designer's causal understanding of the cascading mechanism provides the composition knowledge needed for performing the given design task. The design process itself involves the composition of the functional representation of NAC with the functional representation of the cascading mechanism. This can be accomplished by instantiating the functional representation for the cascading mechanism, and inserting copies of the functional representation of the old SAC into its subfunction slots.

5.4. Design Example 4

It is easy to see that there are many other examples of a similar nature for which the functional representation of the designer's causal understanding of devices integrated with case-based reasoning can be used for design problem solving. An especially interesting example along the lines of the above discussion is the adaptation of the design of NAC to perform the function of heating water. The design and functional representation of NAC may be retrieved from memory for this design problem since the secondary function of NAC is HeatWater and this might best match the specification of the desired device functionality if the designer knows of no other devices of similar functionality. However, adapting the design of NAC for heating water is clearly a very complex design problem since the feedback and feedforward controls are set up wrong. A qualitative simulation of the functional representation of NAC for the primary purpose of heating water would detect the problems with the feedback and feedforward controls. If the designer has available to him functional templates for *generic feedback and feedforward mechanisms* then he may use these templates to correctly set up the feedback and feedforward controls to obtain the design of the water heater.

6. Conclusions

Let us summarize the basic points of our.proposal for organization of knowledge for design problem solving. While case-based reasoning is indeed an attractive approach to design problem solving because of its use of experiential knowledge, an account is needed of how the structure of an existing design is to be adapted to achieve a new device functionality. This capability requires a causal understanding of how the structure of a device enables the accomplishment of its function.

The Functional Representation scheme is a method for organizing and representing the designer's causal understanding of devices. In this scheme, the device functions are used to index the behaviors that compose functional abstractions of structural components into device functions. This organization of knowledge enables a designer to identify the portions of the device structure that need to be modified to achieve a new functionality, and to reason about the effects of these structural changes. The integration of this capability with that of indexing, storing, and retrieving previous design cases provides a powerful strategy for efficiently solving complex design problems.

A knowledge-based system based on this proposal for design problem solving is currently under implementation at our laboratory. The system is intended to address the complex issues of integrating model-based reasoning with case-based reasoning, including the related issues of learning and memory.

Acknowledgments

This research has been supported by the National Science Foundation, grant CBT-87-03745, and the Defense Advanced Research Projects Agency, RADC contract F30602-85-C-0010.

References

1. D. Brown and B. Chandrasekaran, ''Knowledge and Control for a Mechanical Design Expert System'', *IEEE Computer* 19(7):92-100, July 1986.

2. B. Chandrasekaran, J. Josephson and A. Keuneke, ''Functional Representation as a Basis of Generating Explanations'', in *Proceedings of the IEEE Conference on Systems, Man and Cybernetics* Atlanta, October 1986, pages 726-731.

3. B. Chandrasekaran, J. Smith and J. Sticklen, ''Deep Models and their Relation to Diagnosis'', Technical Report, Laboratory for Artificial Intelligence Research, The Ohio State University, June 1987.

4. B. Chandrasekaran, ''Design: An Information Processing-Level Analysis'', Technical Report, Laboratory for Artificial Intelligence Research, The Ohio State University, January 1988.

5. P. Friedland, ''Knowledge-based Experiment Design in Molecular Genetics'', PhD. Dissertation, Computer Science Department, Stanford University, 1979.

6. A. Goel and B. Chandrasekaran, ''Understanding Device Feedback'', Technical Report, Laboratory for Artificial Intelligence Research, The Ohio State University, March 1988.

7. K. Hammond, ''Case-based Planning: An Integrated Theory of Planning, Learning and Memory'', PhD. Dissertation, Computer Science Department, Yale University, October 1986.

8. A. Mackworth, ''Consistency in Networks of Relations'', *Artificial Intelligence*, 8(1), 1977.

9. J. McDermott, ''R1: A Rule-based Configurer of Computer Systems'', *Artificial Intelligence*, 19(1) 39-88, 1988.

10. E. Sacerdoti, *A Structure for Plans and Behaviors*, New York: American Elsevier, 1977.

11. R. Schank, *Dynamic Memory: A Theory of Learning in Computers and People*, Cambridge University Press, 1983.

12. V. Sembugamoorthy and B. Chandrasekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems", in *Experience, Memory and Reasoning*, J. Kolodner and C. Riesbeck: editors, Lawrence Erlbaum, Hillsdale, NJ, 1986.

13. M. Stefik, "Planning with Constraints, MOLGEN: Part 1", *Artificial Intelligence* 16(2):141-169, 1981.

14. G. Sussman, "A Computational Model of Skill Acquisition", PhD. Thesis, Mathematics Department, MIT, 1973.

Understanding Device Feedback

Ashok Goel
B. Chandrasekaran

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus
Ohio 43210

(614)-292-1413
goel-a@ohio-state.arpa

Abstract

Reasoning about the behaviors of a device requires, of course, a language for representing the reasoner's understanding of the device. Moreover, reasoning about complex devices computationally efficiently requires a scheme for organizing the reasoner's knowledge of the device behaviors such that they are easily accessible at the needed level of abstraction. In the *functional representation scheme* [5] for expressing a problem solving agent's understanding of a device, the behaviors are organized around the functions of the device and its structural components. In this paper we extend this scheme to express an agent's understanding of feedback and feedforward interactions common in complex devices. We discuss how feedback and feedforward functions lead to nonlinear device behaviors, and the knowledge structures needed to capture these functions and behaviors.

Understanding Device Feedback

Ashok Goel and B. Chandrasekaran

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University

1. Functional Representation

Most research on qualitative reasoning has been focused on predicting and explaining behaviors of physical devices and processes (*e.g.* [3]). Reasoning about the behaviors of a device requires, of course, a language for representing the reasoner's understanding of the device. Moreover, reasoning about complex devices computationally efficiently requires a scheme for organizing the reasoner's knowledge of the device behaviors such that they are easily accessible at the needed level of abstraction. In relation to this, Sembugamoorthy and Chandrasekaran [5] have proposed that a problem solving agent's knowledge of device behaviors may be organized around higher level abstractions such as the functions of the device and its structural components. In their *functional representation scheme* an agent's understanding of a device is expressed as hierarchically organized schemata, in which the nodes are the intrinsic functions of the device and its components, and the arcs are the behaviors that result in the accomplishment of these functions. The behaviors themselves are represented as acyclic directed graphs, in which the vertices are partial states of the device, and the edges are causal state transitions.

The central thesis of this scheme is that problem solving agents often understand the functioning of a complex device by decomposing the device function into the functions of its structural components. The functioning of a component is similarly understood in terms of the functions of its subcomponents. This decomposition may go on upto as many levels as needed, with only limited interactions between a few components at any level. In the recomposition phase, the functions of the components are composed by behaviors to obtain the function of the device. The function of a device component is similarly obtained by behaviors that compose the functions of its subcomponents. The specification of a behavior at any level may include pointers to deeper knowledge and assumptions underlying the recomposition at that level.

The functional representation scheme has been used for constructing deep models of how problem solving agents understand causal phenomena such as the functioning of simple physical devices [5] and the behaviors of plans viewed as abstract devices [1]. These deep models in turn have been used for qualitative reasoning about the functions and behaviors of various devices, most extensively in the diagnosis of malfunctioning devices [2]. Our aim in this paper is to extend the functional representation scheme to express a problem solving agent's understanding of feedback and feedforward interactions common in complex devices. We will discuss how feedback and feedforward functions lead to nonlinear device behaviors, and the knowledge structures needed to capture these functions and behaviors.

2. Structure of Feedback

Let us consider the Nitric Acid Cooler (NAC), a device commonly used in chemical processing plants, for illustrating feedback and feedforward interactions. The mechanical circuit for (a simplified version of) NAC is shown schematically in Figure 1. Hot Nitric Acid (HNO_3) enters the cooler at p_1 with flow rate R and temperature T_1, and exits at p_4 with the same flow rate and a lower temperature T_2, where p_1, p_2 ... are points in the device space. Similarly, cold water (H_2O) is pumped into the cooler at p_5 with flow rate r_1 and temperature t_1, and exits at p_8 with flow rate r_2 and a higher temperature t_2. Inside the heat exchange chamber heat is transferred from hot Nitric Acid to cold water, thereby cooling Nitric Acid from T_1 to T_2 and heating water from t_1 to t_2. The flow rate R of the inflowing Nitric Acid is measured by a flow sensor, and information about perturbations in its value is communicated to the water pump by a signal c_1 in the wire connecting the sensor and the pump. The pump regulates the rate r_1 at which water flows into the cooler to reflect the perturbations in value of R. This is an example of feedforward control since it is applied before the exchange of heat. Similarly, the temperature T_2 of outflowing Nitric Acid is measured by a temperature sensor, and information about perturbations in its value is communicated to the valve by a signal c_2 in the wire connecting the sensor and the valve. The valve regulates the rate r_2 at which water enters the heat exchange chamber to reflect the perturbations in the value of T_2, and releases excess water. This is an example of feedback control.

We will not devote much space here to the issue of representation of structure except to say that the functional representation language provides p imitives for specifying the device components, the relations between them, and their (device independent) functional abstractions. For instance, the schema for the structure of NAC would specify that the chamber $\{p_7, p_3, p_8, p_2\}$ is a component of NAC, tha. the space enclosed by the chamber includes the space enclosed by the pipe $\{p_2, p_3\}$, and that the functions of the chamber are to contain fluid and transport fluid.

3. Function of Feedback (or Interacting Functions)

The top level decomposition of the functions, in terms of which a problem solving agent may understand the functions of NAC is shown in Figure 2. CoolNitricAcidToT_2 is the primary function of the device, where T_2 is some constant temperature. HeatWater is the secondary function of the device; it is also a side function of CoolNitricAcidToT_2. This captures an agent's understanding that while the intended function of NAC is to cool Nitric Acid, as a side effect of this, water is heated as well. Further, while the intention is to keep the temperature T_2 of outflowing Nitric Acid as steady as possible, the temperature t_2 of outflowing water may vary.

At the next level in the network of Figure 2, SupplyWaterToChamberAtRater_2 is a subfunction (or constituent function) of HeatWater; it is also a supporting function for CoolNitricAcidToT_2, *i.e.* its function is to satisfy the preconditions for the accomplishment of the CoolNitricAcidToT_2 function. Similarly, SupplyNitricAcidToPipeInChamber is a subfunction of CoolNitricAcid and a supporting function of HeatWater. This captures an agent's understanding of the *interaction* between the functions of CoolNitricAcidToT_2 and HeatWater, allowing him to reason that since the subfunction for CoolNitricAcidToT_2 is a supporting function of HeatWater and *vice versa*, the Nitric Acid will get cooled if, and only if, water simultaneously gets heated. Further, this enables the agent to view the role of functions from *multiple perspectives*: SupplyWaterToChamberAtRater_2 is a subfunction from the perspective of achieving HeatWater, but a supporting function from the perspective of accomplishing CoolNitricAcidToT_2.

At the next lower level, the feedback and feedforward functions of

ControlWaterFlowIntoChamber and ControlWaterFlowIntoCooler are similarly understood as supporting functions for SupplyWaterToChamberAtRater_2. Thus, the feedforward and feedback functions are viewed as fulfilling the preconditions for the accomplishment of some higher level function, in this case the SupplyWaterToChamberAtRater_2 function which is itself a supporting function of CoolNitricAcidToT_2

The schemas for some of these functions are shown in Figure 3. The underlined expressions are the primitives of a functional representation language, each with an associated semantics. The primitives Given and ToMake provide an input-output specification of the functions, while By specifies the behavior that results in the accomplishment of the function. Thus each function in the network can be used to *index* the behaviors responsible for accomplishing it. Provided specifies the states of the device in which only a given function can be accomplished, and relates the function to its supporting

4. Behavior of Feedback (or Nonlinear Behaviors)

The directed graphs for the behaviors that achieve some of the NAC functions discussed above are shown in Figure 3. The primitive Using-Function specifies the function of some component that is used by the behavior in accomplishing some higher level function, while By refers to some lower level behavior. The specification of a behavior may include pointers to deeper causal knowledge and assumptions underlying a causal state transition in the behavior. For instance, Behavior1 for accomplishing the function of CoolNitricAcidToT_2 uses (Generic-Knowledge1) that may be stated as follows: In accordance with the Zeroth Law of Thermodynamics, in the context of the Chamber$\{p_7,p_3,p_8,p_2\}$ enclosing the Pipe$\{p_2,p_3\}$, heat will flow from hot Nitric Acid to cold water resulting in a decrease in the temperature of Nitric Acid from T_1 to some T_2, and an increase in the temperature of water from t_1 to some t_2.

Similarly, Behavior1 accomplishes the CoolNitricAcidToT_2 under Assumption1 which may be stated as follows: The relation between temperature T_1 and flow rate R of inflowing Nitric acid, the desired temperature T_2 of outflowing Nitric acid, and the temperature t_1 and flow rate r_2 of water flowing into the heat exchange chamber, is such that the capacity of water to absorb heat in the chamber exceeds the capacity of Nitric Acid to release heat. In essence, the assumption is that the perturbations in the values of the variables T_1 and R are small enough that it is possible to compensate for them by changing the value of the parameter r_2.

The interactions between the functions of a device are, of course, reflected in the behaviors that accomplish the functions. For instance, Behavior1 for accomplishing the function of CoolNitricAcidToT_2, and Behavior2 for achieving HeatWater shown in Figure 3, interact in that Behavior1 will result in cooling Nitric Acid to T_2 if and only if Behavior2 simultaneously results in heating water. This interaction is being captured by the primitive Predicate which specifies that the causal transition from one device state to another in some behavior is conditional on some other device state being true.

We note that the behaviors for accomplishing these interacting device functions are *nonlinear* in the same sense that the plans to achieve interacting goals are often nonlinear [4]. That is, while the device behaviors can be *partially ordered*, each individual behavior being a linear sequence of causal state transitions, a total ordering of the behaviors is typically not possible. Instead, a network of behaviors mirroring the network of Figure 2 collectively results in the functioning of the device. In fact, for the specific case of the skeletal NAC, the device behaviors are inherently *non-serializable.* Thus, if a problem solving agent were to perform a qualitative simulation to verify whether Behavior1 will indeed lead to cooling of Nitric Acid to T_2, then he will have to perform "in parallel" a simulation to check if Behavior2 indeed results in heating water.

5. Understanding Feedback

In our approach, feedback and feedforward are represented as functions that control the values of certain parameters of the device. These control functions are achieved by nonlinear behaviors that communicate information about perturbations in the values of the device variables. The important point here, however, is that reasoning about the functions and behaviors of a complex device can be computationally very expensive, especially in the presence of feedback and feedforward interactions. It is computationally advantageous to organize the understanding of the device into a hierarchical network of functions such that there are only limited interactions between a few functions at any level. During problem solving, when needed these functions can be used to index the individually linear behaviors responsible for accomplishing them.

Representations of devices are there, of course, to be *used*. In fact their use provides the only criterion for judging their adequacy. We have so far used the functional representation of devices primarily for solving two types of problems. In one, when the diagnostic reasoner has incomplete knowledge of certain types, the functional representation can be interpreted and the missing diagnostic knowledge can often be derived. Since the function of the device is represented as being achieved by means of a behavioral sequence, whose causal transitions are ultimately related to the functions of the components, the functional representation yields malfunction hierarchies. Further, since the causal sequences incorporate information about what states fail to result due to malfunctioning of certain components, the representation can also yield observations which may be used to verify malfunction hypotheses. Sticklen [6] has used this idea to develop a diagnostic system, which accesses the functional representation of disease processes for deriving additional diagnostic knowledge.

Another use of the functional representation of a device is to derive qualitative simulations, not from first principles or by using qualitative physics, but by tracing the causal paths organized by functions. Sticken [6] has studied the use of such simulations for examining certain types of interaction between the components of a device. Since the causal sequences are available in stored form and organized functionally, the real work in such simulations is not in the generation of behaviors, but in tracing the effect of certain actions on the functionality of the system.

What functions ought to be included in the representation depends, of course, on the level at which the agent is engaged in problem solving. For instance, if the task is to predict the behavior of a chemical processing plant of which NAC is but one small component, then it is useless to represent the feedback interactions inside NAC. At this level, NAC may best be viewed as a "black box" that operates as a homeostatic device and cools Nitric Acid to a constant temperature. Alternatively, if the task was to explain the functioning of the temperature sensor, then again, it is meaningless to represent feedback interactions at the level of NAC. However, if the task was, say, diagnosis of NAC itself, then a representation of feedback interactions in NAC would be clearly useful. We may add that although we have used NAC as an example to illustrate feedback and feedforward interactions, the functional representation scheme and language that we have used for representing these interactions are device and domain independent and more generally applicable.

6. Functional Representation and Qualitative Simulation

Qualitative simulation is an alternative approach to reasoning about devices in general, and device feedback in particular. In the qualitative simulation method of de Kleer and Brown [3], first the relevant parameters and constraints of the device are determined from its structure and represented as qualitative differential constraints, then a differential perturbation is introduced into the system and a qualitative simulation is performed, and finally changes in the values of the parameters are tracked. There is no *explicit* representation of behavior or function *per se*, instead, the changes in the values of the parameters are first interpreted as behaviors which may then be ascribed a function. de Kleer and Brown have illustrated the use of this method for reasoning about device feedback in an air pressure regulator.

There are several features in common to the method of de Kleer and Brown and our scheme for reàsoning about device feedback. Both approaches view feedback as a function, not as a behavior. More importantly, there is a major emphasis in both approaches on making explicit the (otherwise tacit) assumptions underlying reasoning about devices. There are clearly several differences between the two approaches as well. While their work is more concerned with the *qualitative physics* of device feedback, our primary concern is with a problem solving agent's *cognition* of feedback. Moreover, while their approach is more concerned with the *correctness* of solutions, we are more concerned with the *computational efficiency* of reasoning.

Given a device structure, there is the task c´ [illegible] ȷ its behavior, which is the problem that is attacked by qualitative simulation. However, the agent also needs to organize this behavior in such a way as to explain how the functions of the device are made possible. For simple systems, the distinction between behavior and function is not significant, since relevant behaviors are often also the functions. For complex systems, however, the functions need to be used to index and organize the causal sequences that the structure-to-behavior reasoning has generated. Thus, the functional representation scheme and the qualitative simulation methodology are best viewed as complementary to each other. While the functional representation scheme seeks to capture the *content* of a problem solving agent's understanding of device feedback, the method of qualitative simulation may provide one of the *mechanisms* by which the agent acquires the representation. This relationship between the two approaches works in the other direction as well. For instance, a major drawback of the method of qualitative simulation is that since simulation is global reasoning process, for complex devices the method can be computationally very expensive, especially in the presence of feedback and feedforward interactions. The functional representation scheme, because of its hierarchical nature, may help localize the qualitative simulation to some portion of the device. The integration of the two approaches to form a complete and coherent framework of how problem solving agents understand the functioning of devices, acquire this understanding, and use it for problem solving, however, remains an open research issue.

Acknowledgments

This research has been supported by the Defense Advanced Research Projects Agency, contract RADC F30602-85-C-0010, the Air Force Office of Scientific Research, grant AFOSR 87-0090, the National Heart Lung and Blood Institute, grant HL-38776, and the National Science Foundation, grant CBT-8703745.

References

[1] B. Chandrasekaran, J. Josephson and A. Keuneke. "Functional Representation as a Basis for Generating Explanations". In the *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, Atlanta, Georgia, 1986, pages 726-731.

[2] B. Chandrasekaran, J. Smith and J. Sticklen. "Deep Models and their Relation to Diagnosis". In *Artificial Intelligence in Medicine*, Furukawa (editor), Amsterdam, Netherlands: Science Publishers, 1988 (in press).

[3] J. de Kleer and J. Brown. "A Qualitative Physics Based on Confluences". *Artificial Intelligence*, 24(1984):7-83.

[4] E. Sacerdoti. "The Nonlinear Nature of Plans". In the *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, 1975, pages 206-214.

[5] V. Sembugamoorthy and B. Chandrasekaran. "Functional Representation of Devices and Compilation of Diagnostic Problem Solving Systems". In *Experience, Memory, and Reasoning*, J. Kolodner and C. Riesbeck (editors), Hillsdale, New Jersey: Lawrence Erlbaum, 1986, pages 47-73.

[6] J. Sticklen. "MDX2: An Integrated Medical Diagnostic System". PhD. Dissertation, Department of Computer and Information Science, The Ohio State University, 1987.

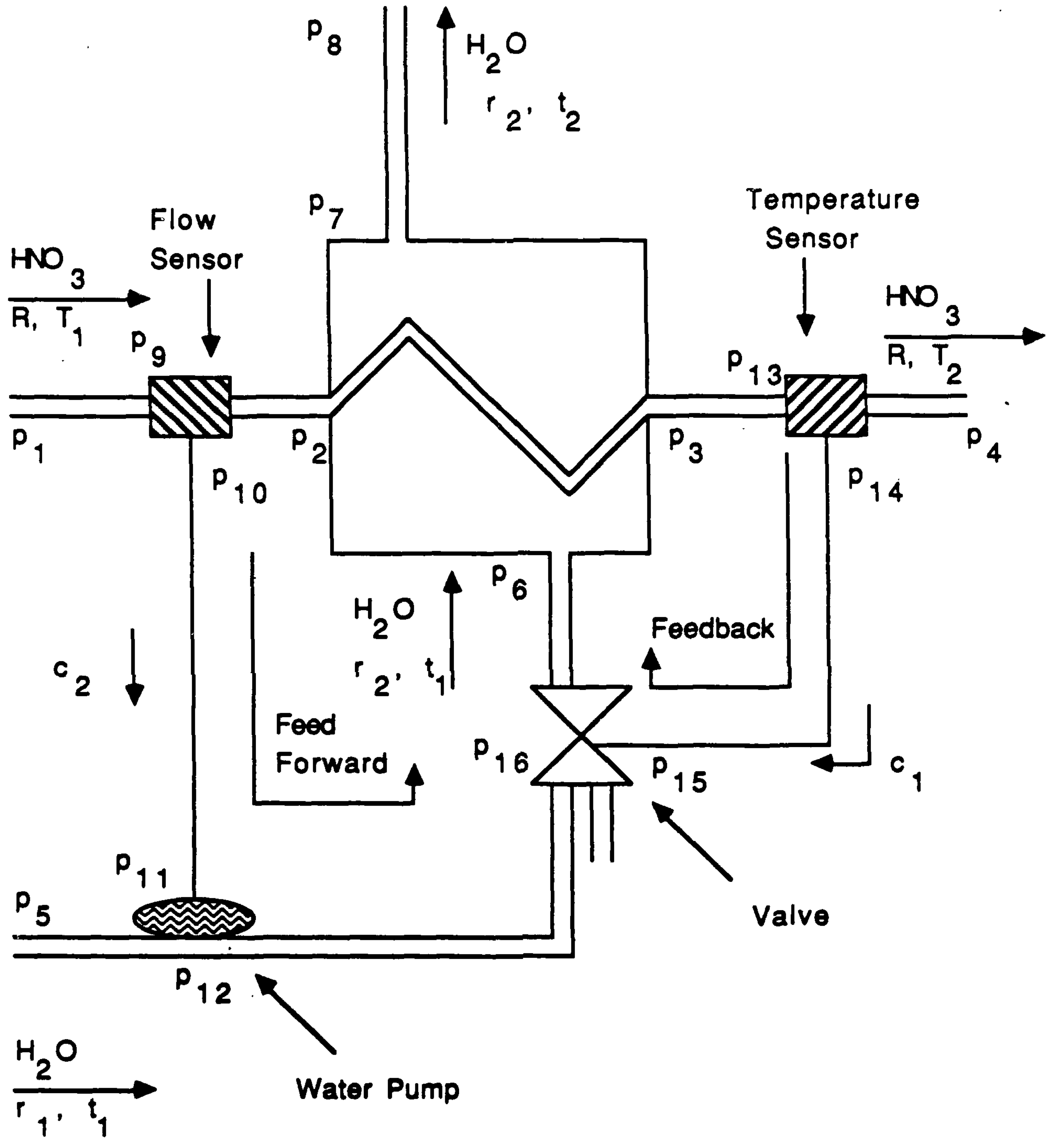


Figure 1 : The Nitric Acid Cooler

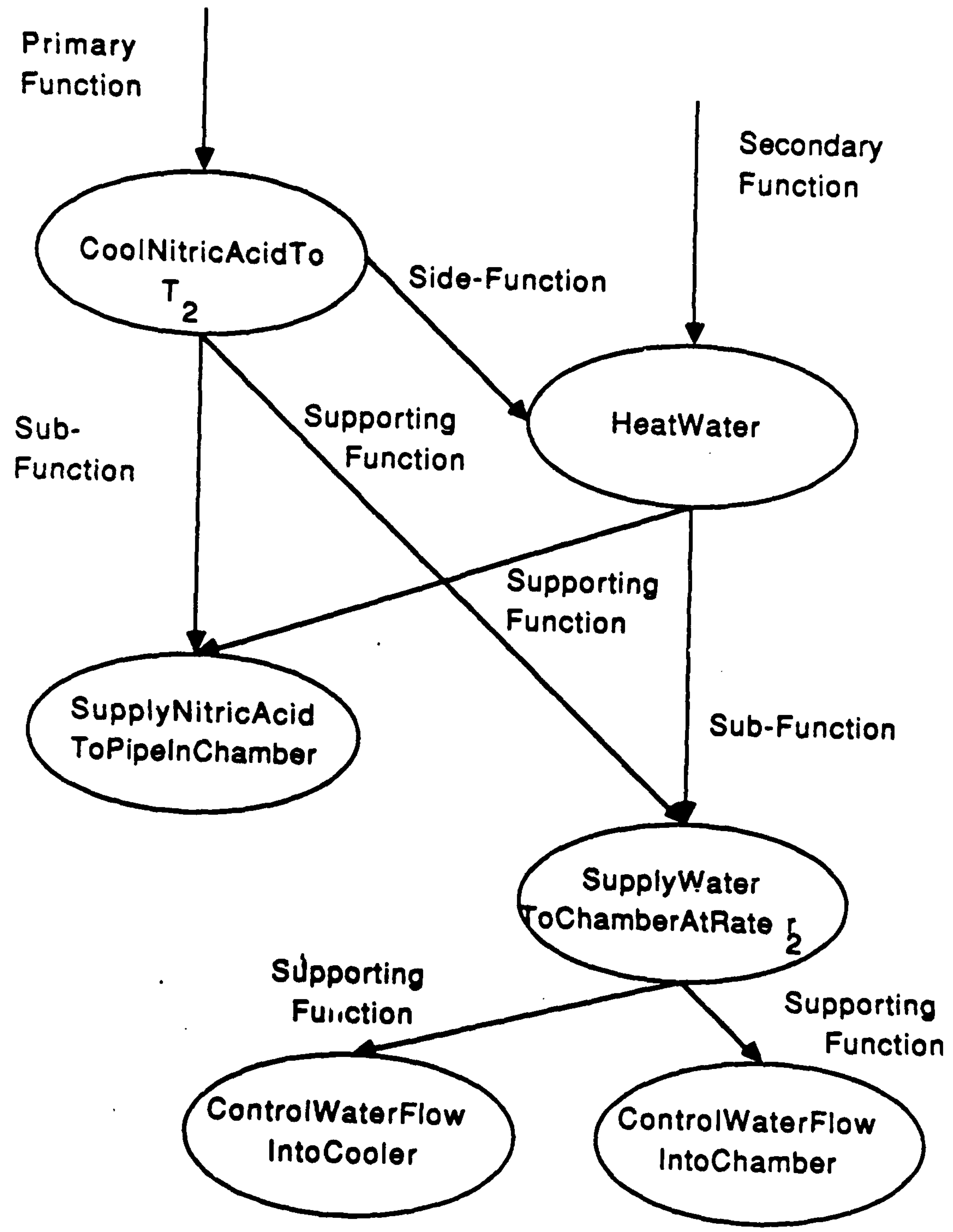


Figure 2a: Functional Organization of the Nitric Acid Cooler

```
Function: CoolNitricAcidTo $T_2$
  Given:
    $HNO_3$ at $p_1$ with
    Flow Rate $R$ and Temperature $T_1$
  ToMake:
    $HNO_3$ at $p_4$ with
    Flow Rate $R$ and Temperature $T_2$
  By: Behavior1
  Provided:
    $H_2O$ at $p_6$ with
    Flow Rate $r_2$ and Temperature $t_1$
End Function CoolNitricAcidTo $T_2$

Behavior1
ToAchieveFunction: CoolNitricAcidTo $T_2$

  $HNO_3$ at $p_1$
  with Flow Rate $R$ and Temperature $T_i$
          !
          ! By: Behavior3
          V
  $HNO_3$ at $p_2$
  with Flow Rate $R$ and Temperature $T_1$
          !
          ! Predicate: $H_2O$ at $p_7$ with
          ! Flow Rate $r_2$ and Temperature $t_1$
          !
          ! As-Per: Generic-Knowledge1
          !
          ! With: Assumption1
          !
          ! Using-Function: Transport Fluid
          ! of Pipe $\{p_2, p_3\}$
          V
  $HNO_3$ at $p_3$
  with Flow Rate $R$ and Temperature $T_2$
          !
          ! Using-Function: Transport Fluid
          ! of Pipe $\{p_3, p_4\}$
          V
  $HNO_3$ at $p_4$
  with Flow Rate $R$ and Temperature $T_2$
End Behavior1
```

Figure 3: Some Functions and Behaviors of NAC

```
Function: HeatWater
   Given:
      $H_2O$ at $p_5$ with Temperature $t_1$
   ToMake:
      $H_2O$ at $p_8$ with
      Flow Rate $r_2$ and at Temperature $t_2$
   By: Behavior2
   Provided:
      $HNO_3$ at $p_2$ with
      Flow Rate $R$ and Temperature $T_1$
End Function HeatWater

Behavior2
ToAchieveFunction: HeatWater

   $H_2O$ at $p_5$ at Temperature $t_1$
            !
            ! By: Behavior4
            V
   $H_2O$ at $p_7$
   with Flow Rate $r_2$ and Temperature $t_1$
            !
            ! Predicate: $HNO_3$ at $p_2$ with
            ! Flow Rate $R$ and at Temperature $T_1$
            !
            ! As-Per: Generic-Knowledge1
            !
            ! Using-Function: Transport Fluid of
            ! Chamber$\{p_7, p_3, p_8, p_2\}$
            V
   $H_2O$ at $p_7$
   with Flow Rate $r$ and Temperature $t_2$
            !
            ! Using-Function: Transport Fluid of
            ! Pipe $\{p_7, p_8\}$
            V
   $H_2O$ at $p_8$
   with Flow Rate $r$ and Temperature $t_2$
End Behavior2
```

Figure 3(continued): Some Functions and Behaviors of NAC

```
Function: SupplyWaterToChamberAtRate$r_2$
   Given: $H_2O$ at $p_5$ at Temperature $t_1$
   ToMake: $H_2O$ at $p_7$ with
            Flow Rate $r_2$ and Temperature $t_1$
   By: Behavior4
   Provided:
      (i) Control Signal $c_1$ at $p_{11}$
      (ii) Control Signal $c_2$ at $p_{15}$
End Function SupplyWaterToChamberAtRate$r_2$

Behavior4
ToAchieveFunction: SupplyWaterToChamberAtRate$r_2$

   $H_2O$ at $p_5$ at Temperature $t_1$
            !
            ! Predicate: Control Signal $c_1$
            !  at $p_{16}$
            !
            ! Using-Function: Pump $H_2O$ of
            ! WaterPump
            V
   $H_2O$ at $p_{12}$ with
   Flow Rate $r_1$ and Temperature $t_1$
            !
            ! Using-Function: Transport Fluid
            ! of Pipe {$p_{12}$,$p_{15}$}
            V
   $H_2O$ at $p_{15}$ with
   Flow Rate $r_1$ at Temperature $t_1$
            !
            ! Predicate: Control Signal $c_2$
            ! at $p_{15}$
            !
            ! By: Behavior7
            V
   $H_2O$ at $p_7$ with
   Flow Rate $r_2$ at Temperature $t_1$
End Behavior4
```

Figure 3(continued): Some Functions and Behaviors of NAC

```
Function: ControlWaterFlowIntoChamber
   Given: $HNO_3$ at $p_{13}$
           with Flow Rate $R$ and Temperature $T_2$
    ToMake: Control Signal $c_2$ at $p_{15}$
    By: Behavior6
End Function ControlWaterFlowIntoChamber


Behavior6
ToAchieveFunction: ControlWaterFlowIntoChamber

   $HNO_3$ at $p_{13}$
   with Flow Rate $R$ and at Temperature $T_2$
         !
         ! Using-Function: Measure Temperature
         ! of Temperature Sensor
         V
   Control Signal $c_2$ at $p_{14}$
         !
         ! Using-Function: Transmit Signal of
         ! Wire $\{p_{14}, p_{15}\}$
         V
   Control Signal $c_2$ at $p_{15}$
End Behavior6
```

Figure 3(continued): Some Functions and Behaviors of NAC

Towards a Neural Architecture for Abductive Reasoning

Ashok Goel
J. Ramanujam
P. Sadayappan

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210, USA

(614)-292-1413
goel-a@osu20.ircc.ohio-state.edu

August 5, 1988

An earlier version of this paper appears in the *Proceedings of the Second IEEE International Conference on Neural Networks*, San Diego, California, July 24 - 27, 1988, Volume [1], pages 681-688.

Towards a Neural Architecture for Abductive Reasoning

Ashok Goel, J. Ramanujam and P. Sadayappan

Department of Computer and Information Science
The Ohio State University

Abstract

The general information processing task of abduction is to infer a hypothesis that best explains a set of data. A typical subtask of this is to synthesize a composite hypothesis that best explains the entire data from elementary hypotheses that can explain portions of the data. The synthesis subtask of abduction is computationally very expensive, more so in the presence of certain types of interactions between the elementary hypotheses. In this paper, we first formulate the general task of abduction as a constrained optimization problem that is nonlinear and nonmonotonic. We then consider a linear monotonic version of the problem and present a neural network for solving it. The neurons in this network represent the elementary hypotheses and the connections between them are symmetric. We find that the energy function for representing the abductive problem contains product forms, and that the minimization of this function requires a network of order greater than two. We then propose a second order network which is composed of functional modules that reflect the structure of the abductive problem. In this architecture, the constraints of the problem are represented explicitly and the connections between the neurons are asymmetric. We suggest how this model can be extended to solve the general task of abduction.

1 Abductive Inference

Abduction is the very general information processing task of inferring a causal hypothesis that best explains a given set of data [4]. Abduction occurs, for instance, in diagnostic problem solving, where the data is in the form of manifestations (or symptoms) and the causal explanatory hypotheses are about malfunctions (or diseases) [14]. Scientific data interpretation, where the data is in the form of sensor readings and the hypotheses are about object structures, and strategic situation assessment, where the data is in the form of field events and the hypotheses are plans ascribed to the adversary, are also instances of abductive inference making. Some aspects of language understanding, image processing, and speech recognition appear to be abductive in character as well.

A typical subtask of abduction is *classification* of the given data into potentially relevant elementary hypotheses stored in memory [11]. The hypotheses are matched with the data, and depending on the degree of match, a *prima facie* belief value for each explanatorily relevant hypothesis is determined. For simple abductive problems, for instance diagnosis under the single fault assumption, the classification subtask often yields hypotheses that can individually explain the entire data. For such problems, the elementary hypothesis with the highest belief value represents the best explanation.

In general, however, an elementary hypothesis that can explain the entire data may not be available. Instead, a composite hypothesis has to be *synthesized* from elementary hypotheses that can explain various portions of the data. A composite hypothesis is operationally the best explanation for the data if it explains as much of the data as possible, if it is made up of most plausible elementary hypotheses, and if it is parsimonious. Synthesizing a composite hypothesis that satisfies these criteria for a best explanation is computationally very expensive, more so in the presence of certain types of interactions between the elementary hypotheses [1]. This has led to some attempts at exploiting concurrency in synthesizing composite hypotheses (*e.g.* [12]), including our own work on concurrent synthesis of composite hypotheses on a shared memory architecture [6], and on a distributed memory message passing architecture [8]. Our work on distributed synthesis of composite hypotheses has quite naturally led us to think in terms of neural architectures for the task. From a parallel processing viewpoint, neural networks form an attractive proposal for an efficient, fine grained, massively parallel machine dedicated to some special class of problems.

In this paper we first formulate the general abductive task as a constrained optimization problem that is nonlinear and nonmonotonic. We then consider a linear monotonic version of the problem and discuss two different neural architectures for solving it. Finally, we suggest how this model can be extended to solve the general task of abduction problem. We note that a connectionist architecture for diagnostic problem solving viewed as an instance of abductive reasoning has recently been reported by Peng and Reggia [13]. However, their model for the task of abduction is significantly different from our model. Their architecture for solving the abductive problem is also substantially different from the architectures proposed in this paper.

2 Abduction as a Combinatorial Optimization Problem

2.1 Characterization of the Task

Let $D = \{d_i | i = 1, \ldots, N\}$ be a finite set of N data (effects, facts). Let $H = \{h_j | j = 1, \ldots, M\}$ be a finite set of M elementary (causal) hypotheses. Let q be a map from subsets of H to subsets of D; $q : 2^H \rightarrow 2^D$. We interpret $q(h_j) = D_j$, where $h_j \in H$ and $D_j \subseteq D$, as the *explanatory coverage* of h_j, *i.e.* h_j can explain only and all members of D_j.

Let $D_o = \{d_i | i = 1, \ldots, n\}$ be a set of n *observed* effects and *given* facts, where $D_o \subseteq D$. The information processing task of abduction may be characterized as a five-tuple $< D, H, q, D_o, H_c >$, where D, H, q, and D_o are the inputs to the task, and H_c, the output of the task, is a subset of H_e, $H_c \subseteq H_e$, that best explains D_o.

Let $B = \{b_k | k = 1, \ldots, l\}$ be a finite set of l belief values. Let $H_e \subseteq H$ such that each $h_j \in H_e$ can explain some non-empty subset of D_o. Let p be a map from H_e to B; $p : H_e \rightarrow B$. We interpret $p(h_j)$ as the *prima facie* belief value for h_j. The classification subtask of abduction takes D, H, D_o, and q^{-1} as input, where q^{-1} is the inverse map of q; and gives H_e and p as output (*cf.*[7]).

Since the classification problem is relatively well understood, and a number of neural architectures for solving it already exist (*e.g.* [2]), we will not discuss it any further in this paper. Instead, we will focus on the synthesis subtask of abduction, which takes D_o, H_e, q, and p as input, and gives H_c as output.

2.2 Characterization of the Best Explanation

The best explanation can be operationally characterized based on the following three optimization criteria:

1. Maximal explanatory coverage of data: A composite hypothesis H_{c1} is a better explanation of D_o than another composite hypothesis H_{c2} if $q(H_{c1}) \supset q(H_{c2})$ [strictly, if $(q(H_{c1}) \cap D_o) \supset (q(H_{c2}) \cap D_o)$]. Ideally, the assembled composite hypothesis, H_c, would provide complete explanatory coverage of D_o, *i.e.*, $q(H_c) = D_o$. This condition represents the constraints on the abduction problem.

2. Maximal belief in hypothesis: A composite hypothesis H_{c1} is a better explanation of D_o than another composite hypothesis H_{c2} if $p(H_{c1}) > p(H_{c2})$. In one model of abduction [8], $p(H_{c1}) > p(H_{c2})$ implies that
$\forall d \in D_o$, and $\forall h_2 \in H_{c2}$ such that $d \in q(h_2)$,
$\exists h_1 \in H_{c1}$ such that $d \in q(h_1) \wedge p(h_1) \geq p(h_2)$.

This specifies that the component hypotheses in H_c should be *locally* optimal in terms of their belief values.

3. Minimal hypothesis: A composite hypothesis H_{c1} is a better explanation of D_o than another composite hypothesis H_{c2} if $|H_{c1}| < |H_{c2}|$. This *global* optimization condition specifies that H_c should be *parsimonious*. In a somewhat different model of abduction [8], this criterion is stated as $H_{c1} \subset H_{c2}$, which leads to an inherently sequential mode of processing.

We note the potential for conflict between these three criteria for a best explanation. This conflict can be resolved by imposing a *precedence relation* according to which maximal coverage of the data has the highest precedence and parsimony of the composite hypothesis has the lowest. We note also that depending on the maps q and p the synthesis task may be underconstrained, in which case the synthesized hypothesis would only be a best explanation. Synthesizing a composite hypothesis that satisfies the above criteria for a best explanation is *NP-Complete* [1].

2.3 Interactions Between Elementary Hypotheses

Several distinct types of interaction are possible between two hypotheses $h_1, h_2 \in H_e$ [11]:

- Associativity: The inclusion of h_1 in H_c suggests the inclusion of h_2. Such an interaction may arise if there is knowledge of, say, a statistical association between h_1 and h_2.
- Additivity: h_1 and h_2 cooperate additively where their explanatory capabilities overlap. This may happen if h_1 and h_2 can separately explain some datum $d \in D_o$ only partially, but collectively can explain it fully.
- Incompatibility: h_1 and h_2 are mutually incompatible, *i.e.*, if one of them is included in H_c then the other should not be included.
- Cancellation: h_1 and h_2 cancel the explanatory capabilities of each other in relation to some $d \in D_o$. For example, h_1 might imply that an increase in some data value, while h_2 may imply a decrease in the value, thus canceling each others explanatory capability with respect to that datum.

The synthesis subtask of abduction is *nonlinear* in the presence of incomptability interactions and *nonmonotonic* in the presence of cancellation interactions.

3 Abductive Inference on a Neural Network

We now consider a special version of the general problem of synthesizing composite hypotheses that is linear, and hence also monotonic. The synthesis task is *linear* [1] if

$$\forall h_i, h_j \in H_e, q(h_i) \cup q(h_j) = q(\{h_i, h_j\})$$

and it is *monotonic* [1] if

$$\forall h_i, h_j \in H_e, q(h_i) \cup q(h_j) \subseteq q(\{h_i, h_j\})$$

In this version of the problem, we assume that the elementary hypotheses are non-interacting, *i.e.* they offer mutually compatible explanatory alternatives where their explanatory coverages overlap. We also assume for simplicity that the belief values found by the classifier for all $h \in H_e$ are equal to 1. (Later, we relax these assumptions and consider the general version of the abductive task).

Under these conditions the task of synthesizing a composite hypothesis can be represented by a bipartite graph consisting of nodes in the set $D_o \cup H_e$. There are no edges between the nodes in D_o, nor are there any edges between the nodes in H_e The edges between the nodes in D_o and the nodes in H_e can be represented by a matrix Q of m rows and n columns, where the rows correspond to data $d_i \in D_o$ and the columns correspond to hypotheses $h_j \in H_e$. The entries in the matrix may be denoted Q_{ij} and indicate whether a given data is explained by a specific hypothesis. The entries are defined as:

$$Q_{ij} = \begin{cases} 0 & \text{if datum } \mathrm{d_i} \text{ is not explained by hypothesis } \mathrm{h_j} \\ 1 & \text{if datum } \mathrm{d_i} \text{ is explained by hypothesis } \mathrm{h_j} \end{cases}$$

Given matrix Q for the bipartite graph, the synthesis subtask of abduction can be modeled as a set-covering problem, which is to find the minimum number of columns that cover all the rows (*cf.* [15]). This ensures that the composite hypothesis will explain all elements of D_o and will be parsimonious (since the belief values for all $h \in H_e$ are assumed to be 1, these are the only two remaining criteria for characterizing a best explanation).

3.1 A Neural Model of Computation

Hopfield and Tank [9,10] have proposed a neural network in which highly interconnected neurons collectively compute good solutions to difficult optimization problems such as the Traveling Salesman Problem (which is *NP-complete*). The neurons in the network are analog devices which may make them closer to biological neurons than strictly digital models. The power of this model of computation comes from the rapidity with which acceptable solutions are found, though the solutions are not guaranteed to be the globally optimal. The emphasis in the model is on exploitation of massive parallelism as opposed to the pursuit of the best solution, the meaning of which often carries a certain degree of arbitrariness for many real world problems.

The processing elements (or neurons) can be modeled as amplifiers having a sigmoid input-output relationship defined by $V_j = g(u_j)$ where the output voltage V_j of amplifier j is a function of the input voltage u_j. Connection between pairs of neurons is defined by a connectivity matrix $\{W_{ij}\} : i, j = 1, \ldots, N$; a negative value of connectivity indicates that the connection is inhibitory. Hopfield [6] has shown that in the case of symmetric connections between neurons *i.e.*, $\forall i \neq j \quad W_{ij} = W_{ji}$, the network evolves to stable states in which the outputs of all neurons remain a constant. The time evolution of the individual neurons is given by

$$du_j/dt = -u_j + \sum_{i=1}^{N} W_{ij} V_i + I_j$$

where I_j represents the input bias current to the j^{th} neuron. These external bias currents serve the purpose of setting the threshold values of the gain functions. The specific values of the steady-state output voltages V_j obtained from the time evolution of the network are determined by the bias currents I_j and the initial values of the input voltages u_j.

If the diagonal elements of the W matrix are zero and the amplifiers operate in a high gain mode, *i.e.*, their gain functions are good approximations to threshold functions, then the stable states of the network are the local minima of the *energy function* defined by

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij} V_i V_j - \sum_{j=1}^{N} V_j I_j$$

The state space of the network of N neurons is defined by the interior of a hypercube of dimension N, and the set of all local minima of energy E is defined by the values of W_{ij}. By a proper choice of the gain function, the local minima of E can be constrained to occur at the corners of the N-dimensional hypercube *i.e.*, with all the values of $V_j = 0$ or 1. This is especially useful when one is interested in digital solutions (0 or 1) to a given problem. An appropriate choice for the gain function is

$$V_j = \frac{1}{2}(1 + \tanh(u_j))$$

3.2 A Neural Network for Abductive Inference

To solve problems using neural networks, one must cast them into the neural network model. For the synthesis subtask of abduction, we associate a neural variable V_j with each hypothesis $h \in H_e$ to indicate if the hypothesis is included in the composite hypothesis C. We minimize

$\Sigma_{j=1}^{m} V_j$ (the cardinality of the composite hypothesis) subject to the constraint that all the data $d \in D_o$ are explained *i.e.*, $\forall i = 1, 2, \ldots, n$, $\Sigma_{j=1}^{m} Q_{ij} V_j \geq 1$. We derive the energy function as follows:

$$E = \alpha * \sum_{j=1}^{m} V_j \quad + \quad \beta * \sum_{i=1}^{n} \prod_{j=1}^{m} \{(1 - Q_{ij}) + (1 - V_j)\}$$

where α and β are positive constants. The first term in the energy function represents the cardinality of the composite hypothesis and the second term represents the penalty for lack of complete coverage. The second term will have a value 0, which is its lowest, in the case of complete coverage - for each datum $d \in D_o$, the product will be 0 if there is at least one hypothesis $h \in H_e$ that can explain it, and among those that can explain it, at least one of them is included in the composite hypothesis. Since, ensuring complete coverage of the data has a higher precedence than parsimony of the composite hypothesis, the constant β should be much larger than α.

We note the appearance of the *product* form in the energy function. This requires a k-th order neural network, where $k = \max_i \sum_{j=1}^{m} Q_{ij}$, the largest number of hypothesis $h \in H_e$ that can explain any one of $d \in D_o$. A k-th order neural network admits up to k-way connections among the neurons [16]. A k- way connection could be thought of as a bus connecting those neurons which participate in the connection. With symmetric k-way connections, the energy function of such a network can be shown to be non-increasing with time. An example of a 3-way symmetric connection is one where

$$\forall i, j, k \qquad W_{ijk} = W_{ikj} = W_{jki} = W_{jik} = W_{kij} = W_{kji}$$

i.e., the connectivity W is the same for all permutations of the subscripts. We note that such a higher order network can solve the synthesis problem using only m neurons, one for each $h \in H_e$.

4 A Functional Architecture for Abductive Inference

4.1 A Structured Neural Network for Abduction

While the notion of a higher order neural network for abductive reasoning appears theoretically attractive and may even admit practical implementations, it is not clear how plausible this model is from the biological and cognitive viewpoints. From the biological perspective, the evidence for the existence of higher order synaptic connections is unclear. From the cognitive perspective, the model is poor in the primitives used to represent the abduction problem. While the causal explanatory hypotheses are represented explicitly in the model, the constraints of explaining the given data are represented only implicitly in the form of the energy function. This implies, among other things, that the capability of the network to explain its actions and justify its conclusions may be rather limited.

These observations have led us to think in terms of a neural model in which the constraints of the abductive problem are represented explicitly. The neural network may be composed of different subnets, each responsible for ensuring the satisfaction of the different criteria for the best explanation. For the linear version of the synthesis problem that we have considered so far in this paper, one subnet may represent the hypotheses $h \in H_e$ and ensure global parsimony of the composite hypothesis, while another subnet may explicitly represent the local constraints of explaining the data elements $d \in D_o$ and ensure maximal explanatory coverage of D_o. These subnets may be viewed as *functional modules* that reflect the structure of the problem and cooperatively perform the task of abduction. We note that assigning different functions to subnets comprising the network makes for a structured neural network [5].

In fact, this is the functional organization that we have used for the distributed memory message passing model for abductive reasoning [8]. In this model, one layer of processes representing the m elementary hypotheses are responsible for ensuring parsimony of the composite hypothesis (as well as for accommodating interactions between the hypotheses for the general abductive problem). Another layer of processes corresponding to the n data elements are responsible for ensuring maximal coverage of data (as well as ensuring maximal belief in the composite hypothesis for the general abductive problem). Each process representing an explanatory hypothesis makes local decisions based on the knowledge available to it and communicates its results to the relevant data processes. Similarly, each process corresponding to a datum to be explained makes local choices and communicates its results to the appropriate hypotheses processes. (For the general abductive problem, the hypotheses' processes also communicate with one another to accommodate interactions and ensure consistency of the growing composite hypothesis). This flow of information back and forth between the two layers of processes continues until a composite hypothesis is fully synthesized. We note that a similar functional organization of processes that allows feedback of information between two layers of processes has been used in neural models based on the *adaptive resonance theory* [2]. However, neural architectures based on this theory have so far been used only for the task of classification.

Tank and Hopfield [17] have proposed a neural network for the linear programming problem that implicitly captures many of these ideas. We are currently developing an integrated neural model for the general problem of abduction based on their scheme. In this model there are $m + n$ neurons. The m neurons which represent hypotheses $h \in H_e$ in the module responsible for ensuring parsimony are denoted as $f_j, j = 1, \ldots, m$. These neurons operate in the high gain mode in order to provide 0/1 solutions. The module responsible for satisfying the constraints of explaining each $d \in D_o$ is made up of n neurons denoted by $g_i, i = 1, \ldots, n$, one for each constraint, $\Sigma_{j=1}^{m} Q_{ij} V_j \geq 1$. The output of the neuron g_i is zero if the corresponding datum d_i is explained by some elementary hypothesis in the composite hypothesis.

The pattern of connectivity among the modules represents the interaction between the competing criteria which characterize the best explanation. Thus, in the network for abductive inference, the output of each f_j hypothesis neuron is transmitted to the input of those constraint neurons g_i which represent a constraint that the hypothesis can satisfy. Similarly, the output of each g_i neuron is transmitted to those f_j neurons that can help satisfy the constraint. The relative speeds of operation of the neurons in the different modules reflect the precedence relationship among the criteria. Thus the g_i neurons operate at a much higher speed than the f_j neurons reflecting the precedence relation between the criteria of complete coverage and parsimony.

4.2 Extensions to the Model

It is easy to extend the above model for the general synthesis subtask of abduction. The *prima facie* belief values of the elementary hypotheses can be incorporated into the model by assigning different weights to the connections from the f_j neurons to the g_i neurons. Thus the weight of the connections from the f_j neuron representing the elementary hypothesis $h \in H$ may represent the belief value $p(h)$.

The model can also accommodate certain types of interactions between the elementary hypotheses. In fact, the mechanism for handling additive interaction between two hypotheses $h_1, h_2 \in H_e$ is implicit in the architecture. The associative interaction can be accommodated by providing an excitatory connection between the neurons representing the two hypotheses. Similarly, the incompatibility interaction between two hypotheses can be accommodated by providing an inhibitory connection between the neurons corresponding to them. However, we know of no easy way to accommodate the cancellation interaction between two hypotheses; this is not particularly disturbing since cancellation is poorly understood in general, and has not yet been adequately captured in any model for abductive inference, neural or otherwise.

This functional neural architecture for abductive inference, unlike the previous model, requires neither symmetric connections between the neurons nor a higher order network. The use of asymmetric connections is closer to the current models of synaptic connections. The reduction in the complexity of the architecture is due to the functional modularization and explicit representation of problem constraints. The computational advantages of explicit representation of abstractions (such as the problem constraints) and functional organization of processing (such as the modularization of the network into specific subnets) are independent of the underlying architecture of their realization. In fact, it is these abstractions and organizations that provide the computational theory of performing a given information processing task such as that of abduction [3].

Acknowledgments

This paper has benefited from discussions with John Hopfield and John Josephson. This research has been partially supported by the National Science Foundation, grant CBT-87-03745.

References

[1] D. Allemang, T. Bylander, M. Tanner and J. Josephson, "On the Computational Complexity of Hypothesis Assembly," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Milan, Italy, August 1987, pages 1112-1117.

[2] G. Carpenter and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," *IEEE Computer* 21(3):77-88, March 1988.

[3] B. Chandrasekaran, A. Goel and D. Allemang, "Connectionism and Information Processing Abstractions," in *Proceedings of the AAAI Symposium on Parallel Models of Intelligence: How Can Slow Components Think So Fast?* Stanford University, Palo Alto, California, March 1988, pages 66-85.

[4] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence,* Reading, MA: Addison-Wesley, 1985.

[5] J. Feldman, M. Fanty, N. Goddard and K. Lynne, "Computing with Structured Connectionist Networks", *CACM* 31(2): 170-187, February 1988.

[6] A. Goel, J. Josephson and P. Sadayappan, "Concurrency in Abductive Reasoning," in *Proceedings of the DARPA Workshop on Knowledge-Based Systems,* St. Louis, Missouri, April 1987, pages 86-92.

[7] A. Goel, N. Soundararajan and B. Chandrasekaran, "Complexity in Classificatory Reasoning," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, July 1987, pages 421-425.

[8] A. Goel, P. Sadayappan and J. Josephson, "Concurrent Synthesis of Composite Explanatory Hypotheses," to appear in *Proceedings of the Seventeenth International Conference on Parallel Processing,* St. Charles, Illinois, August 15-19, 1988.

[9] J. Hopfield, "Neurons with Graded Response have Collective Computational Properties like those of Two-State Neurons," in *Proceedings of the National Academy of Sciences, USA* 81:3088-3092, 1984.

[10] J. Hopfield and D. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics,* 52:141-152, 1985.

[11] J. Josephson, B. Chandrasekaran, J. Smith and M. Tanner, "A Mechanism for Forming Composite Explanatory Hypotheses," *IEEE Transactions on Systems, Man, and Cybernetics,* 17(3):445-454, May/June 1987.

[12] J. Pearl, "Distributed Revision of Composite Beliefs," *Artificial Intelligence* 33(2):173-215, October 1987.

[13] Y. Peng and J. Reggia, "A Connectionist Model for Diagnostic Problem Solving," Technical Report, Department of Computer Science, University of Maryland, November 1987.

[14] H. Pople, "The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning," in *Proceedings of the Fifth International Joint Conference in Artificial Intelligence*, Cambridge MA, August 1977, pages 1030-1037.

[15] J. Reggia, D. Nau and P. Wang, "Diagnostic Expert Systems Based on a Set Covering Model," *International Journal of Man-Machine Studies,* 19:437-460, November 1983.

[16] T. Sejnowski, "Higher-Order Boltzmann Machines," in *Neural Networks for Computing, AIP Conference Proceedings No. 151,* 1986, pages 398-403.

[17] D. Tank and J. Hopfield, "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Transactions on Circuits and Systems,* CAS-33(5):533-541, May 1986.

Generic Tasks and Soar

Todd R. Johnson, Jack W. Smith, Jr., B. Chandrasekaran

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus Ohio, 43210

Abstract

This paper describes our research which is an attempt to retain as many of the advantages as possible of both task-specific architectures and the flexibility and generality of more general problem-solving architectures like Soar. It investigates how task-specific architectures can be constructed in the Soar framework and integrated and used in a flexible manner. The results of our investigation are a preliminary step towards unification of general and task-specific problem solving theories and architectures.

Introduction

Two trends can be discerned in research in problem solving architectures in the last few years: On one hand, interest in ***task-specific architectures*** [Clancey, 1985, Marcus and McDermott, 1987, Chandrasekaran, 1986] has grown, wherein types of problems of general utility are identified, and special architectures that support the development of problem solving systems for those types of problems are proposed. These architectures help in the acquisition and specification of knowledge by providing inference methods that are appropriate for the type of problem. However, knowledge-based systems which use only one type of problem solving method are very brittle, and adding more types of methods requires a principled approach to integrating them in a flexible way.

Contrasting with this trend is the proposal for a flexible, general architecture contained in the work on Soar [Laird et al., 1987]. Soar has features which make it attractive for flexible use of all potentially relevant knowledge or methods. But as a theory Soar does not make commitments to specific types of problem solvers or provide guidance for their construction.

In this paper we investigate how task-specific architectures can be constructed in Soar to retain as many of the advantages as possible of both approaches. We will be using examples from the Generic Task (GT) approach for building knowledge-based systems in our discussion since this approach had its genesis at our Laboratory where it has further been developed and applied for a number of problems; however the ideas are applicable to other task-specific approaches as well.

The GT Paradigm

The GT paradigm is a theory of types of goals and the problem solving methods needed to achieve each type. By problem solving method we mean the specification of behavior to achieve a goal. The paradigm has three main parts:

1. The problem solving of an intelligent agent can be characterized by generic types of goals. Many problems can be solved using some combination of these types.

2. For each type of goal there are one or more problem solving methods, any one of which can potentially be used to achieve the goal.

3. Each problem solving method requires certain kinds of knowledge of the task in order to execute. These are called the ***operational demands*** of the method [Laird et al., 1986].

The term ***generic task*** refers to the combination of a type of goal with a problem solving method and the kinds of knowledge needed to use the method. The GT for classification by establish-refine (called the E-R GT) is given as:

Type of Goal Classify a (possibly complex) description of a situation as a class in a set of categories. An instance of this goal is the classification of a medical case description as one of a set of diseases.

Problem Solving Method This is a hierarchical classification method that works by creating and testing hypotheses about the plausibility that the description of the situation represents an instance of one or more of the classes.

1. If there are no initial hypotheses about what class the description is an instance of, then try to suggest at least one.
2. Try to confirm or reject any hypothesis that is suggested.
3. If a hypothesis is confirmed, determine the possible refinements of the hypothesis and suggest them.
4. If the goal is not met, go to step 2.

Kinds of Knowledge These consist of a refinement hierarchy, hypotheses about the presence of classes, confirmation/rule-out knowledge for these hypotheses, and knowledge to determine when the goal of classification has been achieved.

In addition to classification by establish-refine, GT's have been created for pattern directed hypothesis matching [Johnson et al., 1988], object synthesis by plan selection and refinement [Brown and Chandrasekaran, 1985], and assembly of explanatory hypotheses [Josephson et al., 1987].

GT Systems

A specialized architecture or shell has been constructed for each GT. Each architecture is a combination of an inference engine with a knowledge base. The inference engine is a procedural representation of a GT's problem solving method. The knowledge base provides primitives for encoding the domain specific knowledge needed to instantiate the procedure. We refer to the combination of the encoding of the domain knowledge in the knowledge base and the method that can use it as a problem-solver.

This system building approach offers a number of advantages: First, it is easy to decide when a GT architecture can be used because the knowledge operationally demanded by the method is explicit in the definition of the GT. Second, knowledge acquisition is facilitated because the representational primitives of the knowledge base directly correspond to the kinds of domain knowledge that must be gathered. Third, explanation based on a run-time trace can be couched in terms of the method and knowledge being used to apply it.

Problems with GT Systems

Many flexibility problems arise because a GT architecture contains assumptions not present in the original GT problem solving method. For example, our architecture for hierarchical classification assumes that hypotheses are generated from a pre-defined hierarchy. While this is a common way to generate refinements, other ways exist and might be useful in certain domains. Second, the architecture immediately generates refinements for a confirmed hypothesis. An alternative is to test all the hypotheses in the current state before refining any that were confirmed. Third, the architecture assumes that any problem solver it calls will correctly function. We cannot easily modify the architecture to gracefully handle these situations.

Another set of problems involves the integration of multiple GT problem solvers. The simplest kind of integration is when one problem solver calls on another as a direct means to achieve a subgoal. This is easily done using our current architectures by directly invoking the method and domain knowledge needed to achieve a subgoal. However, sometimes we require more interaction between the problem solvers. For example, in our medical diagnosis systems the hypothesis assembly problem solver has knowledge about those diseases that can occur together and those that are mutually exclusive. This knowledge can be used help guide the classification of diseases; however, it is difficult to implement because the classification architecture has no place for representing or using this knowledge. Our only solution was to specially modify both architectures so that they could interact in the desired way.

Finally, new methods are difficult to add to existing problem solvers; each problem solver must be modified to recognize and use a new method. We would like to have the system automatically consider methods based on the type of goal a method is designed to achieve.

How can Soar Help?

In Soar, all problem solving is viewed as search for a goal state in a problem space. Operators are used to move from state to state. Knowledge in the form of productions is used to select problem spaces, states, and operators. Productions generate *preferences* for an object (ie. a problem space, state, or operator) that indicate how the object relates to the current situation or other objects. Whenever the directly available knowledge is insufficient to make progress Soar automatically generates a subgoal. Therefore, every decision that needs to be made can become a goal to be achieved by searching a problem space. This is called *universal subgoaling.* In knowledge lean situations Soar can make progress by using an appropriate weak method. The weak methods are not explicitly programmed in Soar, but arise from the knowledge available to solve a problem. If the processing in a subgoal is no longer needed, Soar will automatically terminate the subgoal and resume problem solving in a higher level goal. This is called *automatic goal termination.*

Each of these features directly relate to one or more of the limitations with GT systems. The selection of alternatives via preferences allows new options and knowledge to be easily added to existing systems. Brittleness is decreased because of Soar's ability to automatically create subgoals to overcome failures and its ability to fall back on weak methods. Finally, automatic goal termination eliminates unnecessary computation and provides a more natural flow of control.

Mapping GT's to Soar

We have begun to map GT's to the Soar architecture in a straight-forward manner. Each GT is implemented as a problem space; the states represent the developing solution and the operators and operator suggestion rules implement the problem solving method. The required kinds of knowledge can either be represented directly by productions or generated at run-time using additional problem spaces.

To illustrate, we present ER-Soar, an implementation of the E-R GT in Soar. We use a single problem space with three operators: suggest-initial-hypotheses, establish, and generate-refinements.

State Representation The state contains those hypotheses that have been considered and those that are worth immediately considering. Any hypotheses in the state that are refinements of other hypotheses (also in the state) are linked together to form a refinement hierarchy. Each hypothesis also has an indication of whether it has been confirmed, rejected, or not yet judged, and whether it has been refined or not.

Operators The classify problem-space uses 3 operators:

suggest-initial-hypotheses Determine one or more initial hypotheses.

establish <hyp> Determine whether the hypothesis, <hyp>, should be confirmed or rejected.

generate-refinements <hyp> Generate (add to the state) those hypotheses that should be considered as a refinement of <hyp>.

Operator Instantiation An establish operator is created for any hypothesis in the current state that has not yet been judged. A generate-refinements operator is created for any hypothesis that is confirmed but not refined. A suggest-initial-hypotheses operator is created if there are no hypotheses in the current state.

Domain knowledge To use the E-R strategy in a particular domain, knowledge to perform the following functions must be added to the Soar implementation.

- Create the initial state containing one or more initial hypotheses.
- Detect when classification is complete.
- Implement the three operators.

Operator Implementation There are many ways to implement the operators used in the classify space. To make ER-Soar easy to use we have implemented a method for generating refinements from a pre-defined hierarchy and a method for establishing hypotheses based on a confidence value.

Discussion

ER-Soar combines the advantages of the GT approach with the advantages of the Soar architecture. Knowledge acquisition, ease of use, and explanation are all facilitated in ER-Soar because subgoals of the problem solving method and the kinds of knowledge needed to use the method are explicitly represented in the implementation. The subgoals of the method are directly represented as problem space operators. The kinds of knowledge needed to use the method are either encoded in productions or computed in a subgoal. The same advantages apply to the supplied methods for achieving subgoals. Finally, the implementation mirrors the GT specification quite closely making ER-Soar easy to understand and use.

ER-Soar overcomes many of the problems suffered by previous GT systems. Automatic subgoaling allows unanticipated situations to be detected and handled. If no specific method for handling the situation is available, an appropriate weak method can be used. Whenever a goal needs to be achieved it is done by first suggesting problem-spaces and then selecting one to use. This allows new methods in the form of problem-spaces to be easily added to existing problem solvers. If no specific technique exists to determine which method to use, Soar will try to pick one using a weak method. Automatic goal termination provides an integration functionality not available in previous GT architectures. In general, the integration capabilities of ER-Soar are greatly enhanced. Because of preferences and the additive nature of productions, new knowledge can be added to integrate ER-Soar with other methods without modifying existing control knowledge.

Conclusion

ER-Soar illustrates how the advantages of task-specific architectures can be combined with the advantages of a general architecture. The approach used to create ER-Soar is simple and can easily be applied to other task-specific architectures. We are currently using this approach to create Soar versions of the GT's for hypothesis matching and hypothesis assembly. Following this, we will investigate various ways of integrating the three methods.

Acknowledgements

This research is supported by National Heart Lung and Blood Institute grant HL-38776, Air Force Office of Scientific Research grant 82-0255, and Defense Advanced Research Projects Agency, RADC Contract F30602-85-C-0010. Dr. Jack W. Smith, Jr. is supported by National Library of Medicine Career Development Award LM-00083. Computer facilities were enhanced by gifts from Xerox Corporation.

References

[Brown and Chandrasekaran, 1985] Brown, D. C. and Chandrasekaran, B. (1985). Plan selection in design problem-solving. In *Proc. of AISB85 Conference*, Warwick, England. The Society for AI and the Simulation of Behavior (AISB).

[Chandrasekaran, 1986] Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30.

[Clancey, 1985] Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27(3):289–350.

[Johnson et al., 1988] Johnson, T. R., Smith, Jr., J., and Bylander, T. (1988). HYPER: Hypothesis matching using compiled knowledge. Technical report, Lab. for AI Research, CIS Dept., The Ohio State University, Columbus, Ohio.

[Josephson et al., 1987] Josephson, J. R., Chandrasekaran, B., Smith, J. W., and Tanner, M. C. (1987). A mechanism for forming composite explanatory hypotheses. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(3):445–454.

[Laird et al., 1986] Laird, J., Rosenbloom, P., and Newell, A. (1986). *Universal Subgoaling and Chunking*. Kluwer Academic Publishers, Massachusets.

[Laird et al., 1987] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.

[Marcus and McDermott, 1987] Marcus, S. and McDermott, J. (1987). Salt: A knowledge acquisition tool for propose and revise systems. Technical report, Carnegie-Mellon University.

87-JJ-EXHYPO

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
August 1987

A MECHANISM FOR FORMING COMPOSITE EXPLANATORY HYPOTHESES

J. Josephson, B. Chandrasekaran, J. Smith and M. Tanner
Laboratory for Artificial Intelligence Research
Computer Science Department
The Ohio State University
Columbus, OH 43210

This is a revised version of the article sppearing in *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning,* May/June, 1987, pps. 445-454.

Table of Contents

A Mechanism for Forming Composite Explanatory Hypotheses

John R. Josephson, B. Chandrasekaran,
Jack W. Smith Jr., and Michael C. Tanner

Laboratory for AI Research (LAIR)
Department of Computer and Information Science, and
Laboratory for Knowledge Based Medical Systems
Department of Pathology
The Ohio State University
Columbus, Ohio 43210

Contact: John R. Josephson
Ohio State University
LAIR, CIS Dept.
228 CAE Bldg.
Columbus, Ohio 43210-1277

Phone: 614-292-0208
Netmail: Josephson@Ohio-State.ARPA

A revised version of the article appearing in *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Causal and Strategic Aspects of Diagnostic Reasoning*, May/June, 1987, pp. 445-454.

A Mechanism for Forming Composite Explanatory Hypotheses[a]

John R. Josephson, B. Chandrasekaran,
Jack W. Smith Jr., and Michael C. Tanner

Abstract

We describe a general problem solving mechanism that is especially suited for performing a particular form of *abductive inference*, or best explanation finding. A problem solver embodying this mechanism *synthesizes composite hypotheses* by combining simple hypotheses to satisfy explanatory goals. These simple hypotheses are formed by instantiating prestored explanatory "concepts". In this way the problem solver is able to arrive at complex, integrated conclusions which are not pre-stored. We present a computationally-feasible, task-specific problem-solving mechanism for a particular information processing task which is nevertheless of very great generality. The task is that of synthesizing coherent composite explanatory hypotheses based upon a prestored, and possibly vast collection of hypothesis-generating concepts. This is seemingly a common task of intelligence, and potentially a major component of diagnostic reasoning, especially where single-fault assumptions are inappropriate. This work contributes to showing how it is computationally possible to come to "know" based upon the evidence of the case.

In this paper we describe the mechanism both functionally and structurally; that is, the *why* and *what* of the main computations are described, together with algorithms that show *how* each of these computations can be accomplished. The mechanism integrates a classification machine, used for selecting plausible hypotheses, with a specialized means-ends machine, used for assembling a best explanation from the plausible hypotheses thus selected and for pointedly investigating alternative explanations. There are also two other specialized mechanisms for the subsidiary functions of: recognizing the applicability of a hypothesis to the situation, and of interpreting the situation-specific raw data to satisfy the informational needs of the other components. The result of combining these distinct computational mechanisms is an integrated knowledge-based problem solver, functionally suited to its abstract information processing task.

[a]The present paper is an expanded and revised version of "Abduction by Classification and Assembly" which was presented at The Philosophy of Science Association Biennial Meeting for 1986 and appears in PSA 1986, Volume One.

Although the mechanism we describe here is abstracted from the architecture of the Red-2 system, several other diagnostic AI systems embody it too, in varying degrees.

1. Introduction

1.1. Methodology

Artificial Intelligence is the study of complex information-processing problems that often have their roots in some aspect of biological information processing. The goal of the subject is to identify interesting and solvable information processing problems, and solve them. [MARR77] ....

Vision is an information processing task, and like any other, it needs understanding at two levels. The first, which I call the computational theory of an information processing task, is concerned with what is being computed and why; and the second level, that at which particular algorithms are designed, with how the computation is to be carried out. [MARR79]

– David Marr

Cognitive problem solving too can be understood in terms of recognizable information processing tasks subject to the same two-level understanding that is suggested for vision. The computational theory of a cognitive task is an understanding of *what* is being computed (input/output), and *why* (what its significance is); that is, an understanding of the *function* or *goal* the computation. The second level is an understanding of *how* the computation can be carried out by particular algorithms or mechanisms. In this paper we describe one such task, a form of abductive inference. We show how it can be accomplished under appropriate circumstances by way of certain subtasks. We describe the major subtasks both functionally, and in terms of mechanisms whereby they can be efficiently accomplished.

The work reported here takes place in the context of a theory of **generic tasks** in knowledge-based problem solving [CHAN86A]. The theory proposes that complex reasoning processes can be analyzed in terms of a small set of *basic types of reasoning* (the generic tasks) each of which corresponds to an information processing strategy especially suited for achieving a particular knowledge-level functionality. Thus for example hierarchically organized symbolic pattern matching is a particularly apt strategy for **concept matching**, that is, for matching a prestored concept to a situation to determine whether the concept applies to the situation. Generic tasks may be thought of as "types of problem solving [Chan83]", or as "primitive abilities" providing "building blocks of intelligence".

In our laboratory at Ohio State we have developed a software tool for each of the tasks that we have identified so far, and an integrated toolset is presently under development. The generic tasks include: **hierarchical classification** [Bylander86c, Gomez84] **plan selection** and

refinement [Brown, Brown85, Herman] (for routine design and planning), **concept matching** [Bylander86c, Chan86a] **knowledge-directed indirect inference** [Mitt84] (for intelligent data abstraction and retrieval), **prediction by abstracting state changes** [Ronnie], and **assembly and criticism of composite explanatory hypotheses** [Punch].

1.2. Scope

The mechanism described in this paper is that of a composite problem solver which arrives at abductive conclusions by using hierarchical classification, concept matching, knowledge-directed indirect inference, and assembly and criticism of composite explanatory hypotheses. A classification machine, used as a source of plausible hypotheses, is united with a specialized means-ends machine, which is used for assembling an overall best explanation from the plausible hypotheses, and also for criticizing the hypothesis by pointedly investigating the space of alternative hypothesis assemblies. There are also two other specialized mechanisms for the subsidiary functions of: recognizing the applicability of an explanatory concept to the situation, and for interpreting the situation-specific raw data to satisfy the information needs of the other components. The result of putting together these distinct computational mechanisms is an integrated knowledge-based problem solver, functionally suited to the abstract information processing task of forming composite best explanations, based on prestored explanatory concepts.

The present paper will develop the idea of abductive inference, and describe the mechanism and its rationale in some detail. The emphasis will be on the assembly and criticism of composite hypotheses, and on classification as a mechanism for accessing explanatory concepts.

1.3. Red

Red is a knowledge-based medical expert system for use in blood banks as a red-cell antibody identification consultant. [JOSE84b, JOSE85b, SMIT85] The system has now been through two working versions, and a third is under construction at the time of this writing. The present paper presents an abstract description of the problem solving mechanism of Red-2, the second distinct version of the system. Thus Red-2 serves as a working proof of the realizability of most of the abstract design. An evaluation of Red-2's performance has been made [Smith-20-cases], and shows that the system almost always produces clinically acceptable answers, even in complex cases.

2. Abduction

Abduction or Inference to the Best Explanation is a form of inference that follows a pattern something like this:

> D is a collection of data (facts, observations, givens),
> H explains D (would, if true, explain D),
> No other hypothesis explains D as well as H does.
> \---
> *Therefore, H is correct.*

The strength of an abductive conclusion will in general depend on several factors, including:

- how good H is by itself, independently of considering the alternatives,
- how decisively H surpasses the alternatives,
- how thorough the search was for alternative explanations, and
- pragmatic considerations, including
  - the costs of being wrong and the benefits of being right,
  - how strong the need is to come to a conclusion at all, especially considering the possibility of seeking further evidence before deciding.

Abductions, as we have just characterized them, go from data describing something to an explanatory hypothesis that best accounts for that data.

Notice that calling an inference "abduction" carries with it the idea of its goal: a best explanation. Contrast this with characterizing an inference as "deduction", which carries instead the idea of a constraint that is satisfied: that the inference is guaranteed to be truth-preserving. Since there is no intrinsic incompatibility between explanatory goals and truth-preservation constraints, it is conceivable for there to be *deductive abductions*. In fact, if all of the alternative ways of explaining something are exhaustively enumerated, and all but one of the explanations are decisively eliminated, the overall pattern of inference is deductively valid.

> **'There is no great mystery in this matter', he said, taking the cup of tea which I had poured out for him; 'the facts appear to admit of only one interpretation.' [Doyle4] -- Sherlock Holmes**

Even when they are not deductively valid, abductions, besides being intuitively appealing, can be seen to carry logical force. When we have come up with all of the "plausible" explanations we can find for some body of data, and have found compelling, if not decisive, evidence against all but one, then we have good reasons for accepting that one best explanation. Reasons against the other alternatives have been transformed into reasons for accepting the one. Whether we suspend judgment, or go ahead and accept the indicated explanation with some particular degree of confidence, should properly depend on the factors we have enumerated above.

C. S. Peirce used the term "abduction" for a form of inference close to what we describe here [Peirce-p150ff]. Gilbert Harman and others have written of "inference to the best explanation" for essentially the same pattern [HARMAN-65, ENNIS-68, JOSEPHSON-82]; and xxLycan calls it "the explanatory inference" [Lycan-EV]. Sometimes a distinction has been made between an initial process of coming up with explanatorily useful hypothesis alternatives, and a subsequent process of critical acceptance where a decision is made as to which explanation is best. Often the term "abduction" has been reserved for the initial, hypothesis-originating stage [Peirce-p150ff]. We use the term here for the whole process of inferring **from the data to the best explanation.**

John McCarthy has proposed an inference form called "circumscription" to express the logical leap of assuming that all the objects I know about with property P, are in fact all of the objects that exist which have property P. [McCarthy] The leap of assuming (in effect) that all of the plausible explanations we can find, are in fact all of the plausible explanations, can be seen as an application of circumscription. Since circumscription is obviously not a logical axiom (sometimes a circumscription is a reasonable step to take, and sometimes not) it must be that some factors of the situation determine whether a step which has the form of a circumscription is reasonable or intelligent. We have identified above a number of the factors relevant to a presumption that we have covered for all plausible explanations.

Arguably abduction is itself an epistemologically fundamental form of reasoning, not reducible to deduction, probabilistic induction, or any combination of them. [HARMAN-65, MYDISS-CH3].

Whether or not abductions can be justified on logical grounds, they appear ubitiquous in the un-selfconscious reasonings, interpretations, and perceivings of ordinary life, and in the more critically self aware reasonings upon which scientific theories are based [MYDISS-77ff]. It is a common view that *diagnostic reasoning in general is abduction* [Charn85, Pople73, REGGIA85a]. The idea is that the task of a diagnostic reasoner is to come up with a best explanation for the symptoms, findings for the case which show abnormal values. The explanatory hypotheses appropriate for diagnosis are malfunction hypotheses - typically disease hypotheses for physicians, and broken-part hypotheses for mechanical systems.

The characteristic reasoning processes of fictional detectives has been characterized as abduction [SEBEOK-83]. It has been alleged that there are at least 217 abductions to be found in the Sherlock Holmes canon [TRUZZI-217].

The following example is offered in the spirit of showing that abductive reasoning is quite ordinary and commonsensical.

Joe:	Why are you pulling into the filling station?
Tidmarsh:	Because the gas tank is nearly empty.
Joe:	What makes you think so?
Tidmarsh:	Because the gas gauge indicates nearly empty. Also I have no reason to think that the gauge is broken, and it has been a long time since I filled up the tank.

Under the circumstances, the nearly empty tank is the best available explanation for the gauge indication. Tidmarsh's other remarks can be understood as being directed to ruling out a possible competing explanation (broken gauge) and supporting the plausibility of the preferred explanation.

3. Organizing Concepts, Assembling Hypotheses

In some problem situations abduction can be accomplished by a relatively simple matching mechanism, with no particular concern taken for focusing the control of which explanatory concept to consider, or for controlling the assembly of composite explanations. For example, if there are only a small number of potentially applicable explanatory concepts, and if time and other computational resources are sufficiently abundant, then each hypothesis can be considered explicitly. Of course it is a pretty limited intelligence that can afford to try out each of its ideas explicitly on each situation. The classification mechanism we describe here can be seen as meeting the need for organizing the prestored explanatory concepts, and for controlling access to them. It provides a good mechanism for the job whenever knowledge is available of the right sort to make it go.

If the number of potentially applicable hypotheses is at all large, and if more than one can be correct at the same time, then the combinatorics of the situation will not permit us to have one pre-established pattern for each possible conclusion. One main alternative is to actively construct the abductive conclusion as a combination of sub-hypotheses. Up to 2^n different combined conclusions are made available by assembling from a space of n possible hypotheses. Thus a very large space of possible conclusions can result from a relatively small space of primitive categories. For example the Red-2 system has 54 most-detailed hypothesis categories, giving rise to 2^{54} or more than 10^{16} potential conclusions. (Many of these, however, would not be internally consistent, and so would never be produced by the system. Eliminating inconsistent conclusions still leaves more than 10^{12} possible conclusions.) The mechanism we describe here is capable of efficiently picking out the best combination, even from so large a space.

However access to plausibly applicable concepts is controlled, if knowledge is available so that an accurate confidence evaluation can be made for each concept *in isolation from all of the others*, then a composite abductive conclusion can be simply formed as the conjunction of all those concepts that match above some threshold of confidence. Yet this sort of decisive recognition knowledge is rarely available. At the time that knowledge is compiled for a concept, knowledge for decisive confirmation (so called "pathognomonic evidence") may be unavailable, so that the very strongest that can be determined based on direct evidence is that it is "very likely" that the concept applies to the situation. Psychiatric diagnoses probably tend to be of this sort. More significantly, *at run time* there will often be too little actual data about the situation to rule-in and rule-out decisively based on local match, even if there are potential items that would be decisive if they were available. When the data is too weak to resolve the situation besed only on local matching, taking account of the interactions between hypotheses can significantly contribute to getting more conclusion out of the data. An important non-local way to achieve a high degree of confidence is to explicitly rule out alternative ways of explaining things, so that some of the data have only one possible explanation. When this can be done a small hypothesis will stand out as a best explanation for a portion of the data, and thus be a good candidate for inclusion as part of the composite best explanation for the case. In this way a *small abduction* over a portion of the data is performed in support of the *larger abduction* necessary for solving the whole problem. Besides the role of explanatory alternatives, hypotheses may interact in a number of other ways bearing on acceptance, some of which we will discuss later on. In general our explanatory concepts rarely have the kind of independence required for completely separate evaluation to be a viable method, and some form of active control over the formation of composites will be necessary.

4. The Mechanism

The overall function of an abduction machine can be described as that of producing a "best explanation" for a given set of data. An important side effect should be that information is made available about where there are alternative ways of explaining things, so that this information can be used for critically assessing that proposed best explanation, deciding where and how far to trust it.

We present here a functional description of a particular abstract abduction machine, which we may as well call "the Red inference engine" in honor of the system from which it is abstracted. We include enough detail about the major algorithms to make it clear how they work.

4.1. Task and Subtasks

Suppose that we intend to build a knowledge-based system to capture expertise at a certain abductive task. That is, our system is to take, as input, data of a certain type; and produce, as output, best explanations for a well-defined subset of the input data. Suppose that we are given a large number of potentially applicable hypothesis "concepts" or "frames" to base the system on; and that more than one concept can correctly apply at the same time.

Notice that this is precisely the diagnostic situation a physician must face, where the pre-enumerated hypotheses correspond to known and named diseases, and where multiple diseases are common, especially among the very sick people seen at major hospitals, and among those with unobvious ailments. Notice too that this is the situation confronted by the operator of a chemical processing or nuclear power plant, where a single original malfunction can quickly cascade into a multiple malfunction situation. Notice also that this is (an aspect of) the situation faced by *any intelligent knowledge-using agent* facing a complex, changing world, armed primarily with "concepts" of what is possible, and having the goal of trying to "understand" some part of its experience by forming a "good" composite hypothesis.

Suppose further that interactions of various sorts between the pre-enumerated hypotheses can occur, making it unsatisfactory to just match each separately to the case and accept all those above a certain threshold of confidence.

One way to organize a system for this sort of task, and indeed the organization described here, is to set up separate problem-solving structures for the distinct subtasks of:

- coming up with a relatively small number of "plausible" hypotheses from the much larger number of prestored patterns,
- building a "best" composite hypothesis using these plausible hypotheses as available parts, including testing and improving the "goodness" of the composite.

We will see that this decomposition provides a good way of controlling the potentially explosive combinatorics of the problem.

4.2. Removing Irrelevant and Unlikely Hypotheses

By setting one problem solver to filter the primitive hypotheses, letting through only those plausible for the case, we potentially make a great computational contribution to the problem of finding the best composite. By making only moderate cuts in the number, say n, of hypothesis parts to consider, we can make quite deep cuts in the 2^n composites that can be generated from them. For example if we assume that 63 prestored hypothesis patterns get cut down to 8 which are plausible for the case at hand, we have cut the number of potentially generable composites from 2^{63}, which is the number of grains of rice on the last square of the chessboard in the classical story, or more than 9 quintillion; down to $2^8 = 256$.

The INTERNIST system [MILLER-85] (for diagnosis in internal medicine) can be viewed as doing this sort of hypothesis screening when it considers only the subset of prestored diseases that are "evoked" by the present findings. In this way it screens out those hypothesis which are completely irrelevant for explaining the findings. It cuts the number down even further when it scores the evoked diseases for confidence and only continues to consider those above a certain threshold. This can be seen as a kind of screening out of hypotheses for low likelihood of being correct, likelihood being measured primarily by quality of match to the case data.

The DENDRAL system [Buchanan-69] (for elucidating molecular structure from mass spectrogram and other data) explicitly performs such a screening subtask. During the initial "planning" phase it uses the data provided to it to generate a "BADLIST" of molecular substructures that must not appear in the hypothesized composite structures. This BADLIST is used to constrain the search space during the subsequent enumeration of all possible molecular structures consistent with the constraints. That is, (in the present terms) DENDRAL first rules out certain hypotheses for bad match to the case data, and then generates all possible composite hypotheses consistent with not including the ruled-out ones (and other constraints). We note that DENDRAL devotes a separate problem solver, with its own knowledge structures, to the initial screening task.

In the generalized set covering model of diagnosis and abduction [Reggia83, Reggia85b] a disease is associated with a certain set of findings that it potentially covers (i.e. that it can explain if they are present). The diagnostic task is then viewed as that of generating all possible minimum coverings for a given set of findings, by sets associated with diseases (in order to use this as a basis for further question asking). In expert systems built using this model, a match score is computed for each relevant disease each time new findings are entered for consideration, and match scores are used, when appropriate, as the basis for categorically rejecting disease hypotheses from further consideration.

In the Red inference engine a separate problem-solving structure is devoted to the hypothesis selection subtask. It runs first, as soon as the case is started up, and produces a set of hypotheses, each hypothesis being the result of matching a prestored concept to the case. The hypothesis produced are all explicitly relevant for explaining features of the case, and many potential hypothesis do not appear, having been categorically ruled out. Each hypothesis arrives with a symbolic likelihood, the result primarily of case-specific quality of match, but also of case-independent knowledge about frequencies of occurrence. The Red engine is distinguished from INTERNIST and the set-covering systems in that it devotes a separate problem solver explicitly to the hypothesis selection subtask. The advantage of a separate problem solver is that it can be designed specifically for the generic hypothesis selection task.

4.3. Synthesizing a "Best" Composite

Synthesizing a "best" composite can turn out to be computationally expensive. If there are N plausible hypotheses, then there is a space of 2^N composites that can be made from them. If each needs to be generated separately in order to determine which is best, then things can get rapidly out of hand. Clearly it is normally preferable to adopt strategies that allow us to avoid generating all of the combinations.

Sometimes the problem might be completely dominated by the difficulty of coming up with even one good composite explanation. It can be shown that the problem of coming up with just one consistent composite hypothesis that explains everything, under conditions where many of the hypotheses are incompatible with each other, is NP complete.[b]

The authors of DENDRAL saw its job as that of generating all possible composites that were allowable based upon the previously established constraints on submolecules, and the known case-independent chemical constraints on molecular structure. In contrast INTERNIST terminates (after cycles of questioning, which we ignore for these purposes) when it comes up with its single best composite. The set-covering model generates all possible composites of minimal cardinality, but avoids having to enumerate them explicitly by factoring the combinations down into disjoint sets [illegible] ge[illegible]ors.

The Red engine first generates a single, tentative "best" composite, and then improves it by criticism and suitable adjustment. In order for the the criticism to be accomplished, certain other composite hypotheses are generated, but only a relatively small number of them. Again, the Red engine devotes a distinct problem solver to a distinct task, in this case that of forming a best composite. The initial composite hypothesis is formed to be one which explains all of the data (or that part that needs to be

[b]By reduction to 3-SATISFIABILITY [Garey-79]

explained), which is maximally plausible, or nearly so, and internally consistent.[c]

There are often more than simply computational feasibility considerations involved in a decision to generate just the one best composite instead of generating all composites subject to some constraints. For purposes of action an intelligent agent will typically need a single best estimate as to the nature of the situation, even if it is only a guess, and does not need an enumeration of all possible things it could be. By rapidly coming to a best estimate of the situation, the agent arrives quickly at a state where it is prepared to take action. If it had to enumerate a large number of alternatives, this would not only take longer in the generation of them, it would take longer to figure out what to do next. It is difficult to figure out what to do if proposed actions must try to cover for all of the possibilities.

This is not to deny the possibility of situations where careful and intelligent reasoning requires the generation of all of the *plausible* composites (i.e. those with a significant chance of being true), so that they can all be examined and compared. This might especially be called for where the cost of making a mistake is very high, as in medicine, or where there is plenty of time to think over the alternatives. Of course generating *all* of the plausible composites will be computationally infeasible if there are a lot of plausible fragments to choose from and the situation calls for many-part solutions. Moreover, besides being a computationally expensive strategy, generating all of the alternatives will not typically be necessary, as we will often be able to compare composites *implicitly*, by comparing alternative ways of putting them together. For example in comparing little hypotheses h_1 and h_2, we are implicitly (partially) comparing all composites containing h_1 with those containing h_2.

4.4. Criticism: Testing and Improving the Composite

In general it is important to have some idea of how good a composite is, so that an agent can decide whether to act boldly or be cautious; for example deciding to gather more evidence before taking action. Moreover, some critical assessment is necessary, because, as we said, the appropriate confidence of an abductive conclusion depends in part upon how well it stacks up to the competition. This applies to the evaluation of composite hypotheses no less than it applies to simple ones.

For each of the composite hypotheses it constructs, DENDRAL generates a prediction of how the mass

[c]Note that the findings to be explained are in general a *proper subset* of all of the findings of the case. We might try to explain the patient's symptom, but we won't try to explain his age.

spectrogram should appear. Those hypotheses whose predictions mismatch sufficiently are excluded, and the rest are ranked based upon quality of this match. Again DENDRAL devotes a special knowledge-based problem solver to the task, though it is tuned to predictions based upon molecular fragmentation, and is not domain-independent in character.

INTERNIST and the set-covering model appear not to do anything that corresponds to this sort of criticism. Indeed INTERNIST commits itself irrevocably to each hypothesis in the growing composite before it goes on to decide on the next, and has nothing corresponding to the Red engine's tentative initial assembly. The set covering model builds in the critical criterion of simplicity in the form of a guarantee that the problem solver will produce composites with the minimum cardinality sufficient to account for all of the findings.

4.5. Major Modules

The two major modules of the Red inference engine are:

- a classification machine for selecting plausible hypotheses,
- a specialized means-ends machine for assembling a subset of the plausible hypotheses into a "best" composite explanation. The hypothesis assembler is under the control of an overview critic (described here algorithmically) which uses the assembler, first to produce a tentative initial composite, then repeatedly to explore the space of alternative composites, and then finally to build a finished "best explanation" after the pointed investigation of alternative explanations. This overview critic also does some processing to guarantee that the composite it finally produces is parsimonious, i.e., has no explanatorily superfluous parts.

It is important to note that the second of these modules is usable separately from the first one, and so, for example, some structural model of a device could be exploited in some fashion to generate malfunction hypotheses. [DeKleer, Moorthy, GENE84] What the assembler/critic needs is a source of hypotheses, each hypothesis offering to explain some portion of the data, and each evaluated to determine a degree of plausibility. The assembler/critic will also need to have access to various sorts of information about how the hypotheses interact.

4.6. The Classification Machine

Taking the MDX [chandra79a, chandra83a] system as it's point of departure, the classifier is implemented as a taxonomic hierarchy of hypothesis specialists. Each specialist in the hierarchy specializes in a single "concept". When invoked it will match that concept to the details of the case, either ruling it out of further consideration, or else producing a hypothesis that has an associated symbolic likelihood, and offers to explain certain of the findings of the case.

The hierarchy organizes the specialists from most general at the top, to most specific at the tip nodes. The hypothesis selection activity proceeds in a top-down, more-general-to-more-refined manner, taking advantage of the search pruning effect that comes from ruling out whole subtrees of hypotheses by ruling out at high levels of generality. This top-down, prune-or-pursue control regime, associated with MDX-like diagnostic systems has been called "establish-refine". It can in principle proceed in parallel, matching of two sub-concepts being typically independent of each other. By efficiently pruning the search for plausible hypotheses, establish-refine is a significant contributor to taming the combinatorics of the problem space. It makes it efficient and practical to search a very large space of stored concepts for just those that plausibly apply to the case.

4.7. Plausible Hypotheses

Each concept that is considered and cannot be ruled out is matched against the data of the case to produce a description of which parts of the data it can explain (or contribute to explaining), and how plausible it is under the circumstances. Concept matching for plausibility has been discussed elsewhere. [Bylander86c, Chan86a] Each plausible hypothesis delivered by the classifier thus comes with:

- a description, particularized to the case, of which findings it offers to explain,
- a symbolic plausibility value representing a symbolic *prima facie* estimate of likelihood for the hypothesis.

Each plausible hypothesis has its own consistent little story to tell, and to contribute to the larger story representing the abductive conclusion.

4.8. Hypothesis Interactions

Hypothesis interactions are considered to be of two general types, each with its own kind of significance for the problem-solving:

- *explanatory interactions*, e.g., due to overlapping in what the hypotheses can account for, and
- *substantive interactions* of mutual support and incompatibility, e.g., resulting from causal or logical relations.

For example, two disease hypotheses might offer to explain the same findings without being especially compatible or incompatible causally, logically, or definitionally. On the other hand hypotheses might be mutually exclusive (e.g. because they represent distinct sub-types of the same disease), or mutually supportive (e.g. because they are causally associated). The INTERNIST system did not make a clear distinction between hypotheses which are competitors because they are both capable of explaining the same findings in the case (thus not both needed), and those that are competitors because they are mutually exclusive. [Pople77] INTERNIST was really only concerned with the former type. In general the elements of a diagnostic differential need to be *exhaustive* of the possibilities, so that at least one must be correct, but they need not be mutually exclusive. If they are exhaustive then evidence against one of them is transformed into evidence in favor of the others.

The following types of hypothesis interaction can be accommodated and treated appropriately by the mechanism we are describing. Appropriate handling for the first four of them has been implemented and tested:

- A and B are mutually compatible, and represent explanatory alternatives where their explanatory capabilities overlap.
- Hypothesis A is a subhypothesis of B (i.e., a more detailed refinement).
- A and B are mutually incompatible.
- A and B cooperate additively where they overlap in what they can account for.
- Using A as part of an explanation suggests using B also.
- A, if it is accepted, raises explanatory questions of its own that can be resolved by appeal to B.

An example of this last type occurs when we hypothesize the presence of a certain pathophysiological state to explain certain symptoms, and then hypothesize some more remote cause to account for the pathophysiological state. The tummy ache is explained by the presence of the ulcers, and the ulcers are in turn explained by the anxiety neurosis.

4.9. The Hypothesis Assembler

A mechanism for hypotheses assembly is used which is reminiscent of the means-ends regime of GPS, The General Problem Solver. [Newell63] Its overall goal is to explain all of the finding that need to be explained. It detects differences between the goal state (everything explained) and the present state (the working hypothesis does not explain everything), and focuses on a salient difference (a most significant unexplained finding). It uses this unexplained finding to select a hypothesis part to integrate into the growing working composite.

We begin by describing a basic hypothesis assembler, capable only of treating one type of hypothesis interaction. Then we will describe how it can be enhanced to treat the other types of interaction appropriately.

4.9.1. The Basic Assembler

The basic assembler treats only hypotheses that are mutually compatible, and that represent explanatory alternatives where their explanatory capabilities overlap. A set of findings is given, the goal is to assemble an explanation for them, and to do so in a manner that respects the plausibilities of the candidate parts. It works by using the plausibilities to guide the means-ends search.

Procedure:

- Set the initial composite to the empty set.
- Loop until there is nothing left to explain, or nothing left that can be explained.
    - Focus attention on an unexplained finding (initially the whole set is unexplained). If domain knowledge is available to point out the most significant unexplained finding, then well and good; but if not, then the choice can be made at random.
    - Pick the most plausible hypothesis that explains that finding. If no plausible explanation for it can be found, then note the finding as unexplainable and loop again, else continue. If two or more explanations tie for maximal plausibility, choose one at random.
    - Include the newly chosen hypothesis into the set of hypotheses that constitutes the growing composite hypothesis. That is, set the new composite to the union of the old composite and the set whose member is the chosen hypothesis.
    - Compute what the composite can now explain, and determine the unexplained remainder.
- End loop.
- Return the value of the composite.

The basic assembler produces a composite hypothesis which is as complete as possible. Since it uses the most plausible explanatory hypothesis at each choice point, the composite hypothesis is maximally plausible as well, or nearly so.[d]

It is easy and computationally inexpensive to rid the composite of explanatorily superfluous parts.[e] This can be done after the composite is built, or else parsimony can be enforced as the assembly proceeds. We thus arrive at a composite hypothesis which is *as complete as possible, maximally plausible* (or nearly), and *parsimonious.*

[d]The conditions under which this process produces an optimally plausible composite have been investigated, and will possibly form the subject of a future paper. Yet in general this mechanism will deliver the "correct answer" if one stands out on the basis of the evidence, and will fail only if the evidence of the case is weak or erroneous, or if knowledge in the system is wrong or missing. *Thus guarantees of maximal plausibility for the composite are not really very significant.* Intelligent agents do not need to be especially good at making forced choices between nearly equal alternatives.

[e]Check through the parts in order of least plausible to most plausible; for each part compute the explanatory capabilities with the part removed; and check to see if there is any loss.

Note that this interpretation of Occham's Razor has clear epistemic virtues. Logically the composite hypothesis is a conjunction of little hypotheses; so, if we remove one of the conjuncts the resulting hypothesis is distinctly more likely to be true, since it makes fewer commitments. Superfluous hypothesis parts make factual commitments, expose themselves to falsity, with no compensating gain in explanatory power. Thus the sense of parsimony we propose here is such that *the more parsimonious hypothesis is more likely to be true.*

Since the assembly process added monotonically to a growing hypothesis, with incrementally growing explanatory power, and with no backtracking, the process is computationally very inexpensive, at least if it is inexpensive to compute explanatory power. In general the greatest computational expense will be in checking through the available hypotheses to determine which one is the most plausible way to explain the finding of attention. But the classifier will collaborate to reduce the alternatives to a relatively small number, and one pass through the set will suffice. *Thus the whole process of assembly is computationally very efficient.* [ALLE87]

4.9.2. Extensions and Elaborations to the Basic Assembler

Extensions can be made to the basic assembler to handle the other types of hypothesis interaction we have mentioned.

If hypotheses in the space come with subtype relationships, as they normally would with a hierarchical classifier, the assembler can preferentially pursue the goal of explanatory completeness and secondarily pursue the goal of refining the constituent hypotheses down to the level of most detail.

A more difficult problem is in devising a strategy for when some of the hypotheses in the space are mutually incompatible.[f] What was done in Red-2 is to maintain the consistency of the growing hypothesis as the assembly proceeds. If a finding is encountered whose only available maximally plausible explainers are incompatible with something already present in the growing hypothesis, then one of these newly encountered hypotheses is included in the growing hypothesis, removing from the composite any parts inconsistent with the new one.[g] The basic idea is that the finding must be explained, even if that

[f]We assume the ability to detect that hypotheses are incompatible.

[g]If we remove parts from the growing hypothesis we introduce the danger of an infinite loop, but fortunately this can be dealt with fairly readily. We suitably raise the standards for reintroducing a hypothesis for the second time in precisely the same situation in which it was first introduced. The second time around we require that there be no net loss of explanatory power for the whole assembly resulting from reintroducing the hypothesis and removing its contraries from the assembly. There are a variety of acceptable measures of explanatory power that will serve here to guarantee progress.

forces a serious revision of the growing hypothesis. This seems to be a rather weak way of handling incompatibles, and the authors feel that significant improvements are possible.

If hypotheses can cooperate additively where they overlap in what they can explain, all we need to do is to suitably incorporate this knowledge into the methods for computing what a composite hypothesis can explain.

In order to handle the kind of hypothesis interaction where one hypothesis suggests the use of another, as for example if there is available knowledge of a statistical association, we can give extra plausibility credit to the suggested hypothesis if the hypothesis making the suggestion is already part of the growing composite. The availability of a way to grow the hypothesis preferentially along lines of statistical association provides a rudimentary ability for it to grow along causal lines as well.

A more interesting ability to grow along causal lines results if we permit one hypothesis, if it is accepted into the growing hypothesis, to raise explanatory needs of its own. For example, a newly added hypothesis can be posted as a kind of higher-level finding which needs to be explained in its turn by the growing assembly. Thus at the same time that the newly added hypothesis succeeds in explaining some of the findings, it introduces a "loose end". This provides a way in which the growing hypothesis can move from hypotheses close to the findings of the case, and towards more and more remote causes of those findings.

4.10. The Overview Critic

Procedure:

- The assembler is invoked to produce a tentative best explanation.
- Explanatorily superfluous parts are removed.
- The assembler is invoked repeatedly as necessary to assess which of the hypotheses in the composite are **indispensable**. A hypothesis is judged indispensable if removing it from a composite which is a complete explanation leaves behind a composite which cannot then be assembled to completion without reintroducing the removed one. It follows that a hypothesis is indispensable if and only if something that it explains *has no other plausible explanation.* It is important to distinguish between hypothesis parts which are non-superfluous relative to some particular composite, that is they cannot be removed without explanatory loss, and indispensables without which no complete explanation can be found in the whole hypothesis space.
- The non-indispensable parts of the composite are then removed, and the assembler is invoked again to rebuild from the core of indispensables back to a complete explanation. This rebuilding process might again introduce explanatorily superfluous parts that will need to be cleaned out, but it cannot introduce any more indispensables. Since an indispensable explains something that has no other plausible explanation, every indispensable is already

present in any complete explanation.

- Any newly introduced explanatorily superfluous parts are removed.

At the end of this process the composite hypothesis explains as much as possible, is maximally plausible (or nearly so), is parsimonious, and has been built up by going from a core of hypotheses which are *most certain*.

At this stage the best explanation has been *inferred*, or at least A best explanation has been *inferred*, there being no *a priori* guarantee that a best explanation is unique. Under some circumstances the reasoning process will have virtually *proved* the correctness of its conclusion. If each part of the composite is indespensible (in the sense above), then the system has proved that it has produced the correct explanation, assuming that the data is correct and the knowledge base is complete. That is, the system will have proved that it has come up with the only complete and parsimonious explanation available to it. When parts of the conclusion are not indispensible, the system will have discovered that alternative explanations are possible, so appropriate cautions may be taken in using the abductive conclusion.

5. Extensions and Elaborations of the Mechanism

The degree of intimacy, and the nature of the relationships, between the classifier, various critics, and the means-ends assembler, is an unresolved research issue which we are actively exploring. In Red-2, the classifier runs first, producing a set of plausible hypotheses, and then is followed by the critic, which uses the assembler to produce the best explanation. In the future we anticipate a version where the classifier and an assembler/critic run concurrently, with the latter using its perspective on the progress of the problem solving to help guide the search for plausible hypotheses. Red-2's assembler/critic built up composite hypotheses using only tip node hypotheses delivered to it by the classifier. We anticipate that in the next version composites will first be assembled at higher levels of generality, and then refined into more detailed hypotheses using nodes lower in the hierarchy. More distantly we envision a version where lots of little hypothesis assemblers and critics are distributed over a problem solving structure that makes local abductions, producing little assembled best explanations. By solving subproblems the little abducers will serve the needs of larger abducers, and make it possible to assemble hypotheses from parts which are themselves assemblies.

6. Summary

We have described how best explanations can be inferred by a mechanism which tames the combinatorics of very large spaces of explanatory hypotheses. Structured conclusions can be arrived at whose parts are connected by relationships of type-subtype, statistical association, and explainer-explained. An instance of this machine exists, exercising some of the capabilities we attribute to the abstract machine, and arriving at correct answers in complicated situations.

Although the mechanism is an abstraction of the architecture of the Red-2 system, DENDRAL, INTERNIST, and systems based on the Set-Covering model of abduction realize it too, in varying degrees. In this light the present offering should be seen, not so much as contributing new mechanisms, but as showing how existing systems can be analyzed; and how, once we understand the tasks, efficient mechanisms can be devised specifically to achieve them.

More grandly we may say that a computational description has been given to the functional architecture of a possible mind, or rather, of a certain dimension or slice of a possible mind. The kind of synthesis of explanatory hypotheses we describe here is a *generic task of higher intelligence*. It must be accomplished somehow by any intelligent, knowledge-using agent that comes to "know" by calling upon "concepts", attaching them to situations or objects, and using the resulting little hypotheses as materials to form composite "best explanations". The task is general, but specific. There are a limited number of functional architectures that could accomplish it, especially when account must be taken of the constraints imposed by limited knowledge, limited time, and limited computational resources. There are even fewer architectures that are *especially suited* to the task, and we have just described one of them.

7. Acknowledgments

This work has been supported in various stages by NSF Grant MCS-8305032, NIH Grant R01 LM 04298 from the National Library of Medicine, and DARPA/RADC contract F30602-85-C-0010. Dr. Jack W. Smith, Jr. is supported by NLM Career Development Award K04 LM00083. Thanks are due to Tom Bylander for showing that the task of producing a consistent composite is NP complete, and for his helpful comments on a previous draft. Also to Bill Punch and Dean Allemang for their discussions and encouragement of the approach to abduction, to Jon Sticklen for arguing with the ideas until they were better justified, to the members of the recent graduate seminar at Ohio State on diagnostic reasoning for their helpful comments, and to two anonymous reviewers of a previous draft of the paper who, by their

failure to understand what was being presented, pointed the way to an improved explication of the ideas.

References

1. Marr, David, *Artificial Intelligence: A Personal View*, The MIT Press, 1981, Also appears in Artificial Intelligence 9(1):47-48, 1977.

2. Marr, David, *Representing and Computing Visual Information*, The MIT Press, 1979.

3. B. Chandrasekaran and J. Josephson, "Explanation, Problem Solving, and New Generation Tools: A Progress Report", *Proceedings of the Expert Systems Workshop*, April 1986, pp. 122-126.

4. B. Chandrasekaran, "Towards a Taxonomy of Problem-Solving Types", *AI Magazine*,Vol. Winter/Spring, 1983, pp. 9-17.

5. T. Bylander and S. Mittal, "CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling", *AI Magazine*,Vol. 7, No. 3, August 1986, pp. 66-77.

6. F. Gomez and B. Chandrasekaran, *Knowledge Organization and Distribution for Medical Diagnosis*, Addison-Wesley Publishing Company, 1984, pp. 320-338, Also appears in IEEE Transactions on Systems, Man and Cybernetics, SMC-11(1):34-42, January, 1981

7. David C. Brown, "Expert Systems for Design Problem-Solving Using Design Refinement with Plan Selection and Redesign". Ph.D. Dissertation, The Ohio State University

8. D. C. Brown and B. Chandrasekaran, "Plan Selection in Design Problem-Solving", *Proceedings of AISB85 Conference*, The Society for AI and the Simulation of Behavior (AISB), Warwick, Englang, April 1985.

9. D. Herman, J. Josephson, and R. Hartung, "Use of DSPL for the Design of a Mission Planning Assistant", *Expert Systems in Government Symposium*,October 1986, pp. 273-278.

10. S. Mittal, B. Chandrasekaran and J. Sticklen, "PA[illegible]EC: A Knowledge-Directed Data Base for a Diagnostic Expert System", *IEEE Computer Special Issue*,Vol. 17, September 1984, pp. 51-58.

11. Ronnie Sarkar, "State Abstraction Problem Solving". OSU LAIR Technical Report

12. W. F. Punch III, M. C. Tanner and J. R. Josephson, "Design Considerations for PIERCE, a High-Level Language for Hypothesis Assembly", *Expert Systems in Government Symposium*,October 1986, pp. 279-281.

13. Josephson, J. R.; Chandrasekaran, B.; Smith, J.W., "Assembling the Best Explanation", *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, IEEE Computer Society, Denver, Colorado, December 3-4 1984, pp. 185-190, A revised version by the same title is now available as a technical report.

14. J. R. Josephson, M. C. Tanner, J. W. Smith, M.D., J. Svirbely and P. Straum, "Red: Integrating Generic Tasks to Identify Red-Cell Antibodies", *Proceedings of The Expert Systems in Government Symposium*, Kamal N. Karna,ed., IEEE Computer Society Press, 1985, pp. 524-531.

15. Smith, Jack W.; Svirbely, John R.; Evans, Charles A.; Strohm, Pat; Josephson, John R.; Tanner, Mike, "RED: A Red-Cell Antibody Identification Expert Module", *Journal of Medical Systems*,Vol. 9, No. 3, 1985, pp. 121-138.

16. Smith, J., Josephson, J., Tanner,M., Svirbely J., Strohm, P., "Problem Solving in Red Cell Antibody Identification: Red's Performance on 20 Cases", Tech. report, Laboratory for Artificial Intelligence Research, Department of Computer and Information Science, The Ohio State University, 1986.

17. Doyle, Sir Arthur Conan, *The Sign of the Four*, Clarkson N. Potter, Inc., 1960.

18. Peirce, C.S., *Abduction and Induction*, Dover, 1955, pp. 150ff., ch. 11.

19. Gilbert Harman, "The Inference to the Best Explanation", *Philosophical Review*,Vol. LXXIV,

January 1965, pp. 88-95.

20. Ennis, R., "Enumerative Induction and Best Explanation", *The Journal of Philosophy,*Vol. LXV, No. 18, September 1968, pp. 523-529.

21. John R. Josephson, *Explanation and Induction,* PhD dissertation, The Ohio State University, 1982.

22. William G. Lycan, "Epistemic Value", *Synthese,*Vol. 64, 1985, pp. 137-164.

23. John McCarthy, "Circumscription - A Form of Non-Monotonic Reasoning", *Artificial Intelligence,*Vol. 13, 1980, pp. 27-39.

24. John R. Josephson, *Explanation and Induction,* PhD dissertation, The Ohio State University, 1982.

25. John R. Josephson, *Explanation and Induction,* PhD dissertation, The Ohio State University, 1982, pages 77ff

26. Eugene Charniak and Drew McDermott, *Introduction to Artificial Intelligence,* Addison Wesley, 1985.

27. Pople, H., "On the Mechanization of Abductive Logic", *Proceedings of the Third International Joint Conference on Artificial Intelligence,* 1973, pp. 147-152.

28. Reggia, J., "Abductive Inference", *Proceedings of The Expert Systems in Government Symposium,* Kamal N. Karna,ed., IEEE Computer Society Press, 1985, pp. 484-489.

29. Thomas A. Sebeok and Jean Umiker-Sebeok, *"You Know My Method": A Juxtaposition of Charles S. Peirce and Sherlock Holmes,* Indiana University Press, Bloomington, 1983, ch. 2.

30. Marcello Truzzi, *Sherlock Holmes: Applied Social Psychologist,* Indiana University Press, Bloomington, 1983, ch. 2.

31. Miller, R.A., Pople, J.E. Jr. and Myers, J.D., "INTERNIST - I, An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine", *New England Journal of Medicine,*Vol. 307, 1982, pp. 468-476.

32. Buchanan, B. G., Sutherland, G. L. and Feigenbaum, E. A., *Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry,* Edinburgh University Press, Edinburgh, 1969.

33. Reggia, J., "Diagnostic Expert Systems Based on a Set Covering Model", *International Journal of Man-Machine Studies,*Vol. 19, November 1983, pp. 437-460.

34. James A. Reggia, Barry T. Perricone, Dana S. Nau, and Yun Peng, "Answer Justification in Diagnostic Expert Systems--Part 1: Abductive Inference and Its Justification", *IEEE Transactions on Biomedical Engineering,*Vol. BME-32, No. 4, April 1985, pp. 263-267.

35. M. Garey and D. Johnson, *Computers and Intractability,* W. H. Freeman and Company, New York, 1979.

36. Johan de Kleer, "Reasoning About Multiple Faults", *AI Magazine,*Vol. 7, No. 3, August 1986, pp. 132-139.

37. V. Sembugamoorthy and B. Chandrasekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems", *Experience, Memory, and Reasoning,* 1986, pp. 47-73.

38. Genesereth, M.R., "The use of design descriptions in automated diagnosis", *Artificial Intelligence,* Vol. 24, December 1984, pp. 411-436, PA

39. B. Chandrasekaran, F. Gomez, S. Mittal, and J. W. Smith, "An Approach to Medical Diagnosis Based on Conceptual Structures", *Proceedings of the Sixth International Joint Conference on*

Artificial Intelligence, 1979, pp. 134-142.

40. B. Chandrasekaran and S. Mittal, "Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems", in *Advances in Computers*, M. Yovits, ed., Academic Press, Vol. 22, 1983, pp. 217-293.

41. Pople, H., "The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning", *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI 77, 1977, pp. 1030-1037.

42. Newell,A. and Simon,H.A., *GPS, A Program That Simulates Human Thought*, McGraw-Hill, New York, 1963, pp. 279-293.

43. D. Allemang, M. C. Tanner, T. Bylander and J. R. Josephson, "On the Computational Complexity of Hypothesis Assembly", *Proc. Tenth International Joint Conference on Artificial Intelligence*, Milan, August 1987, to appear